INTERNATIONAL STANDARD

ISO/IEC 23003-4

Second edition
2020-06

Information technology — MPEG audio technologies —

Part 4: Dynamic range control

Technologies de l'information — Technologies audio MPEG —

Partie 4: Contrôle de gamme dynamique

Reference number
ISO/IEC 23003-4:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia, and hypermedia*.

This second edition cancels and replaces the first edition (ISO 23003-4:2015), which has been technically revised. It also incorporates the Amendments ISO 23003-4:2015/Amd.1:2017 and ISO 23003-4:2015/Amd.2:2017. The main changes compared to the previous edition are as follows:

— Amendments to the previous edition that include enhancements, definitions of profiles and levels, reference software, and conformance are integrated.

A list of all parts in the ISO/IEC 23003 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

Consumer audio systems and devices are used in a large variety of configurations and acoustical environments. For many of these scenarios, the audio reproduction quality can be improved by appropriate control of content dynamics and loudness.

This document provides a universal dynamic range control tool that supports loudness normalization. The DRC tool offers a bitrate efficient representation of dynamically compressed versions of an audio signal. This is achieved by adding a low-bitrate DRC metadata stream to the audio signal. The DRC tool includes dedicated sections for clipping prevention, ducking, and for generating a fade-in and fade-out to supplement the main dynamic range compression functionality. The DRC effects available at the DRC decoder are generated at the DRC encoder side. At the DRC decoder side, the audio signal may be played back without applying the DRC tool, or an appropriate DRC tool effect is selected and applied based on the given playback scenario.

Loudness normalization is fully integrated with DRC and peak control to avoid clipping. A metadata-controlled equalization tool is provided to compensate for playback scenarios that impact the spectral balance, such as downmix or DRC. Furthermore, the DRC tool supports metadata-based loudness equalization to compensate the effect of playback level changes on the spectral balance.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights

The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC. Information may be obtained from the patent database available at www.iso.org/patents.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those in the patent database. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — MPEG audio technologies —

Part 4:
Dynamic range control

1 Scope

This document specifies technology for loudness and dynamic range control. It is applicable to most MPEG audio technologies. It offers flexible solutions to efficiently support the widespread demand for technologies such as loudness normalization and dynamic range compression for various playback scenarios.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14496-12, *Information technology — Coding of audio-visual objects — Part 12: ISO base media file format*

ISO/IEC 14496-26:2010, *Information technology — Coding of audio-visual objects — Part 26: Audio Conformance*

ISO/IEC 23008-3:2019, *Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 3: 3D audio*

ISO/IEC 23091-3, *Information technology — Coding-independent code points — Part 3: Audio*

3 Terms, definitions and mnemonics

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 14496-12 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1.1
DRC sequence
series of DRC gain values that can be applied to one or more audio channels

3.1.2
DRC set
defined set of DRC sequences that produce a desired effect if applied to the audio signal

3.1.3
album
collection of audio recordings that are mastered in a consistent way

Note 1 to entry: Traditionally, a collection of songs released on a Compact Disk belongs into this category, for example.

3.1.4
conformance test bitstream
bitstream used for testing the conformance of MPEG-D DRC compliant audio decoders

3.1.5
conformance test case
conformance test category and a combination of one or more conformance test conditions for which a conformance test sequence is provided

3.1.6
conformance test condition
condition which applies to properties of a conformance test sequence in order to test a certain functionality of the MPEG-D DRC decoder

3.1.7
conformance test criteria
one or more conformance test tools and corresponding parameters applied to verify the conformance for a certain conformance test sequence

3.1.8
conformance test sequence
set of a conformance test bitstream, a decoder setting, an input audio file and a corresponding reference file

3.1.9
decoder input parameters
input parameters that are supplied to an MPEG-D DRC decoder in addition to a conformance test bitstream, a decoder interface bitstream and an input audio file

3.1.10
decoder setting
combination of a decoder interface bitstream and decoder input parameters that are supplied to an MPEG-D DRC decoder

3.1.11
input DRC set selection parameters
input parameter set for testing of a DRC gain decoder instance

Note 1 to entry: This parameter set is solely used for conformance testing in the context of the DRC gain decoder conformance test category (DrcGainDec).

3.1.12
reference audio file
decoded counterpart of a conformance test bitstream, a decoder setting and an input audio file

3.1.13
reference DRC set selection parameters
decoded counterpart of a conformance test bitstream and a decoder setting fed to the DRC set selection process

Note 1 to entry: This parameter set is an intermediate result of an MPEG-D DRC compliant decoder implementation solely used for conformance testing in the context of the DRC selection process test category (DrcSelProc).

3.1.14
reference file
reference audio file or reference DRC set selection parameters

3.2 Mnemonics

bslbf	bit string, left bit first, where "left" is the order in which bit strings are written in the ISO/IEC 14496 series NOTE Bit strings are written as a string of 1s and 0s within single quote marks, for example '1000 0001'. Blanks within a bit string are for ease of reading and have no significance.
byte_align()	number of bits to fill for byte alignment at the offset of n bits: byte_align(n) = 8 ceil (n/8) – n
uimsbf	unsigned integer, most significant bit first
vlclbf	variable length code, left bit first, where "left" refers to the order in which the variable length codes are written
bit(n)	a bit string with n bits in the same format as bslbf
unsigned int(n)	an unsigned integer with n bits in the same format as uimsbf
signed int(n)	a signed integer with n bits, most significant bit first
mod	modulo operator: (x mod y) = x-y floor (x/y)
sizeof(x)	size operator that returns the bit size of a field x
TRUE/FALSE	values of Boolean data type, which correspond to numerical 1 and 0, respectively

4 Symbols

a_i	filter coefficient
b	band index of DRC filter bank (starting at 0)
b_i	filter coefficient
deltaTmin	smallest permitted DRC gain sample interval in units of the audio sample interval

f_c	cross-over frequency in Hz
$f_{c,norm}$	cross-over frequency expressed as fraction of the audio sample rate
$f_{c,norm,SB}(s)$	cross-over frequency of audio decoder sub-band s expressed as fraction of the audio sample rate NOTE The cross-over frequency is the upper band edge frequency of the sub-band.
f_s	audio sample rate in Hz NOTE If an audio decoder is present, it is the sample rate of the decoded time-domain audio signal.
M_{DRC}	DRC frame size in units of the audio sample interval $1/f_s$
N_{DRC}	maximum permitted number of DRC samples per DRC frame NOTE Identical to the number of intervals with a duration of *deltaTmin* per DRC frame.
N_{Codec}	codec frame size in units of the audio sample interval $1/f_s$
π	ratio of a circle's circumference to its diameter
s	audio decoder sub-band index (starting at 0)
z	complex variable of the z-transform

5 Technical overview

The technology described in this document is called the "DRC tool". It provides efficient control of dynamic range, loudness, and clipping based on metadata generated at the encoder. The decoder can choose to selectively apply the metadata to the audio signal to achieve a desired result. Metadata for dynamic range compression consists of encoded time-varying gain values that can be applied to the audio signal. Hence, the main blocks of the DRC tool include a DRC gain encoder, a DRC gain decoder, a DRC gain modification block, and a DRC gain application block. These blocks are exercised on a frame-by-frame basis during audio processing. In addition to encoded time-varying gain values, the DRC gain decoder can also receive parametric DRC metadata for generation of time-varying gain values at the decoder. Various DRC configurations can be conveyed in a separate bitstream element, such as configurations for a downmix or combined DRCs. The DRC set selection block decides based on the playback scenario and the applicable DRC configurations which DRC gains to apply to the audio signal. Moreover, the DRC tool supports loudness normalization based on loudness metadata.

A typical system for loudness and dynamic range control in the time domain is shown in Figure 1. A more complex system including downmixer and peak limiter is shown in Figure 2. The decoder part of the DRC tool is driven by metadata that efficiently represents the DRC gain samples and parameters for interpolation. The gain samples can be updated as fast as necessary to accurately represent gain changes down to at least 1 ms update intervals. In the following, the decoder part of the DRC tool is referred to as "DRC decoder", which includes everything except the audio decoder and associated bitstream de-multiplexing.

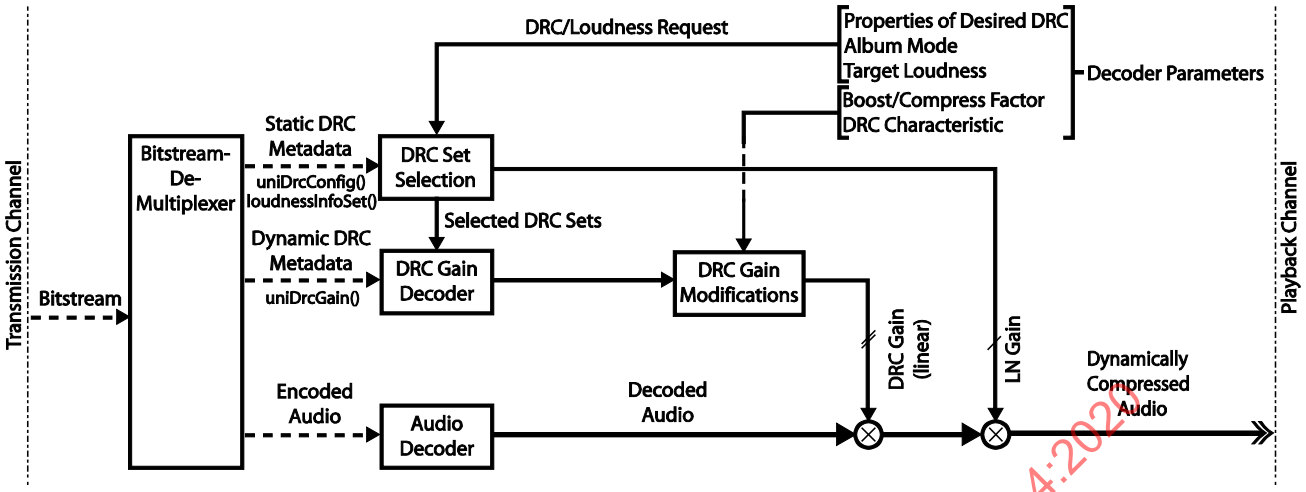


Figure 1 — Block diagram of a typical system with audio decoder and DRC tool modules to achieve loudness normalization (LN) and dynamic range control

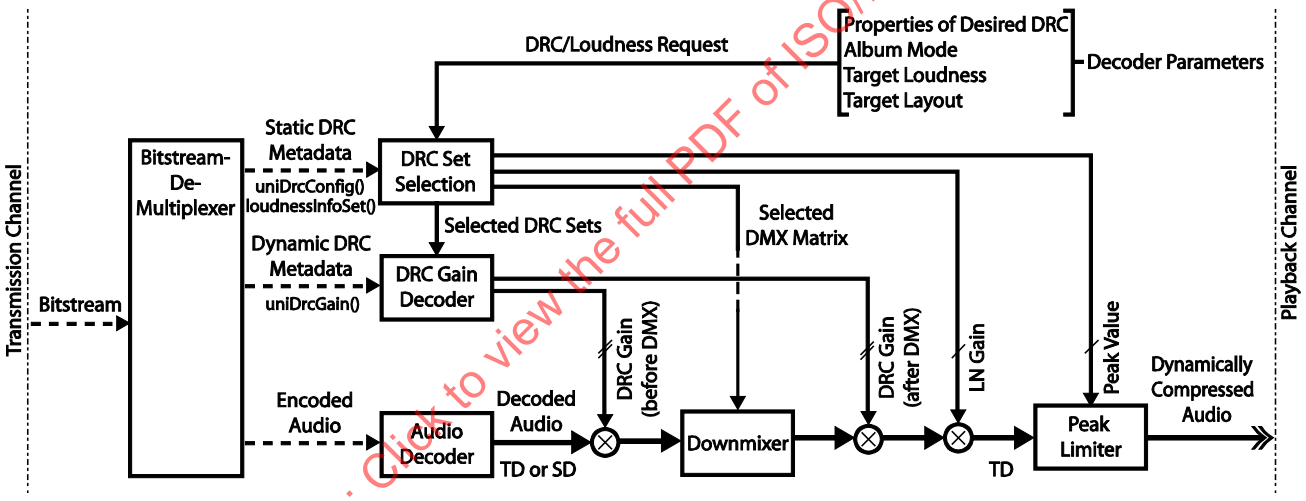


Figure 2 — Block diagram of a more complex system including downmixer and peak limiter (TD = time-domain, SD = subband-domain)

The DRC tool provides support for loudness equalization, sometimes called “loudness compensation”, that can be applied to compensate for the effect of the playback level on the spectral balance. For this purpose, time-varying loudness information can be recovered from DRC gain sequences to dynamically control the compensation module. While the compensation module is out of scope, the interface describes in which frequency ranges the loudness information should be applied.

A flexible tool for generic metadata-controlled equalization is provided. The tool can be used to reach the desired spectral balance of the reproduced audio signal depending on a wide variety of playback scenarios, such as downmix, DRC, or playback room size. It can operate in the sub-band domain of an audio decoder and in the time domain.

The DRC tool is specified in Clause 6. The tool may be subject to profiles and levels that shall be in accordance with Annex I. The bitstream field decoding of the DRC tool shall be in accordance with Annex A. If an interface for external parameter control of the DRC tool is used, it shall conform to Annex B.

6 DRC decoder

6.1 DRC decoder configuration

6.1.1 Overview

The DRC configuration information can be received in-stream using the static payloads uniDrcConfig() and loudnessInfoSet() described below, or it can be delivered by a higher layer, such as in ISO/IEC 14496-12 (see Table 1). The basic decoding process of the static information is virtually the same. The difference consists mainly in a few syntax changes and reduced field sizes to increase the bit rate efficiency of the in-stream configuration. The syntax of the in-stream static payload is given in 7.3. The associated metadata encoding is given in A.6. The static DRC payload is evaluated once at the beginning of the decoding process and it is monitored subsequently. For static DRC payload changes during playback, see 6.12.

Table 1 — Overview of configuration (setup) and separate metadata track in ISO/IEC 14496-12

	Sample entry code	Setup (in sample entry)	Track reference	Sample format
Audio track	As specified for the audio codec in use (unchanged)	DRCInstructions box using negative values for *drcLocation*	"adrc" referring to the metadata tracks carrying gain values	As specified for the audio codec in use (unchanged)
Metadata track	"unid"	(none)	(none)	Each sample is a uniDrcGain() payload

The static payload is divided into several logical blocks:

— channelLayout();

— downmixInstructions(), downmixInstructionsV1();

— drcCoefficientsBasic(), drcCoefficientsUniDrc(), drcCoefficientsUniDrcV1();

— drcInstructionsBasic(), drcInstructionUniDrc(), drcInstructionUniDrcV1();

— loudnessInfo(), loudnessInfoV1();

— drcCoefficientsParametricDrc();

— parametricDrcInstructions();

— loudEqInstructions();

— eqCoefficients();

— eqInstructions().

Except for the channelLayout(), drcCoefficientsParametricDrc(), and eqCoefficients(), multiple instances of a logical block can appear. The DRC decoder combines the information of the matching instances of the logical blocks for a given playback scenario. Matching instances are found by matching several identifiers (labels) contained in the blocks.

From the static payload, the decoder can also extract information about the effect of a particular DRC and various associated loudness information, if present. If multiple DRCs are available, this information can be used to select a particular DRC based on target criteria for dynamics and loudness (see 6.3)

uniDrcConfig() contains all blocks except for the loudnessInfo() blocks which are bundled in loudnessInfoSet(). The last part of the uniDrcConfig() payload can include future extension payloads. In the event that a *uniDrcConfigExtType* value is received that is not equal to UNIDRCCONFEXT_TERM, the DRC tool parser shall read and discard the bits (*otherBit*) of the extension payload. Similarly, the last part of the loudnessInfoSet() payload can include future extension payloads. In the event that a *loudnessInfoSetExtType* value is received that is not equal to UNIDRCLOUDEXT_TERM, the DRC tool parser shall read and discard the bits (*otherBit*) of the extension payload. Each extension payload type in uniDrcConfig() or loudnessInfoSet() shall not appear more than once in the bitstream if not stated otherwise. An extension payload of type UNIDRCCONFEXT_V1 shall preceed an extension payload of type UNIDRCCONFEXT_PARAM_DRC in the bitstream if both payloads are present. For ISO/IEC 14496-12, configuration extension payloads are provided according to Table 76.

The top level fields of uniDrcConfig() include the audio sample rate, which is a fundamental parameter for the decoding process (if not present, the audio sample rate is inherited from the employed audio codec). Moreover, the top level fields of uniDrcConfig() include the number of instances of each of the logical blocks, except for the channelLayout() block which appears only once. The top level fields of loudnessInfoSet() only include the number of loudnessInfo() blocks. The logical blocks are described in the following.

6.1.2 Description of logical blocks

6.1.2.1 channelLayout()

The channelLayout() block includes the channel count of the audio signal in the base layout. It may also include the base layout unless it is specified elsewhere. For use cases where the base audio signal represents objects or other audio content, the base channel count represents the total number of base content channels. The base channel count value shall serve as the value of baseChannelCount for parsing the downmixInstructions(), downmixInstructionsV1(), drcInstructionsUniDrc(), drcInstructionsUniDrcV1() and eqInstructions() payloads as specified in Clause 7.

6.1.2.2 downmixInstructions() and downmixInstructionsV1()

This block includes a unique non-zero downmix identifier (*downmixId*) that can be used externally to refer to this downmix. The *targetChannelCount* specifies the number of channels after downmixing to the target layout. It may also contain downmix coefficients, unless they are specified elsewhere. For use cases where the base audio signal represents objects or other audio content, the *downmixId* can be used to refer to a specific target channel configuration of a present rendering engine. In contrast to downmixInstructions(), the downmixInstructionsV1() payload includes an offset for all downmix coefficients and the coefficient decoding does not depend on the LFE channel assignment. The downmixInstructions() box for ISO/IEC 14496-12 contains the corresponding metadata of either one of the in-stream payloads as indicated by the *version* parameter of the box.

6.1.2.3 drcCoefficientsBasic(), drcCoefficientsUniDrc(), and drcCoefficientsUniDrcV1()

A drcCoefficients block describes all available DRC gain sequences in one location. The block can have the basic format or the uniDrc format. The basic format, drcCoefficientsBasic(), contains a subset of information included in drcCoefficientsUniDrc() that can be used to describe DRCs other than the ones specified in this document. drcCoefficientsUniDrc() contains for each sequence several indicators on how it is encoded, the time resolution, time alignment, the number of DRC sub-bands and corresponding crossover frequencies and DRC characteristics. The crossover frequencies shall increase

with increasing band index. Alternatively, explicit indices in a decoder sub-band domain can be specified for the assignment of DRC sub-bands. The sub-band indices shall also increase with increasing band index. If the DRC gains are applied in the time-domain by using the multi-band DRC filter bank specified in 6.4.12, explicit index signalling is not allowed. The index of the DRC characteristic indicates which compression characteristic was used to produce the gain sequence. The DRC location describes where these gain sequences can be found in the bitstream. The DRC gain sequences in that location are inherently enumerated according to their order of appearance starting with 1.

The DRC location field encoding depends on the audio codec. A codec specification may include this specification, and use values 1 to 4 to refer to codec-specific locations as indicated in Table 2. For example, for AAC (ISO/IEC 14496-3), the codec-specific values of the DRC location field are encoded as shown in Table 3.

Table 2 — Encoding of *drcLocation* for in-stream payload

drcLocation n	Payload
0	*Reserved*
1	Location 1 (Codec-specific use)
2	Location 2 (Codec-specific use)
3	Location 3 (Codec-specific use)
4	Location 4 (Codec-specific use)
$n > 4$	*reserved*

Table 3 — Codec-specific encoding of *drcLocation* for MPEG-4 Audio

drcLocation n	Payload
1	uniDrc() (defined in Clause 7)
2	dyn_rng_sgn[i] / dyn_rng_ctl[i] in dynamic_range_info() (defined in ISO/IEC 14496-3:2009 subpart 4)
3	compression_value in MPEG4_ancillary_data() (defined in ISO/IEC 14496-3:2009/AMD 4:2013)
4	*reserved*

The DRC frame size can optionally be specified. It shall be provided if the DRC frame size deviates from the default size specified in 6.4.2. If not specified, the default frame size is used.

The in-stream drcCoefficient syntax is given in Table 65, Table 67 and Table 68. The syntax for the corresponding block for ISO/IEC 14496-12 (ISO base media file format) is shown in Table 66 and Table 69. The corresponding blocks carry essentially the same information. Values that are identically included in both blocks are coded the same way except for *drcLocation*.

In ISO base media file format (see ISO/IEC 14496-12), for each codec that can be carried in MP4 files and that also carries DRC information, there is a specific definition of how the location is coded, using the *DRC_location* field (see Table 4). A negative value of *DRC_location* indicates that a DRC payload is in an associated meta-data track. That track is the *n*-th linked via a track reference of type "adrc" (audio DRC) from the audio track, where *n* = abs(DRC_location), and the sample-entry type in the meta-data track indicates in which format the coefficients are stored. Table 3 defines the specific entries of the *drcLocation* field for AAC. Some example use cases are discussed in C.10.

If the uniDrc() payload is stored in a separate track in the ISO base media file format (ISO/IEC 14496-12), then the track is a metadata track with the sample entry identifier "unid" (uniDrc), with no required boxes added to the sample entry. The time synchronization with the linked audio track is the same as if the payload was in-stream.

Table 4 — Encoding of drcLocation for ISO/IEC 14496-12

drcLocation *n*	Payload
n < 0	*DRC payload located in |n|-th linked meta-data track*
0	*reserved*
1	Location 1 (Codec-specific use)
2	Location 2 (Codec-specific use)
3	Location 3 (Codec-specific use)
4	Location 4 (Codec-specific use)
n > 4	*reserved*

The drcCoefficientsUniDrcV1() payload is defined in Table 68. It contains the same information as drcCoefficientsUniDrc() except for the assignment of DRC gain sequences to gain sets and the optional specification of a number of parametric DRC characteristics. The drcCoefficientsUniDrc() payload assigns gain sequences in order of transmission. In contrast, the drcCoefficientsUniDrcV1() payload maps a gain sequence by index to *gainSets*. The latter permits to refer to the same gain sequence for multiple DRC bands which is not possible when using drcCoefficientsUniDrc(). If a drcCoefficientsUniDrcV1() payload is present, any drcCoefficientsUniDrc() payload for the same location is ignored.

The drcCoefficientsUniDrcV1() payload can also include information about dynamic equalization filters if the field *shapingFiltersPresent*==1. There can be a number of filters that are indexed in order of appearance. The DRC sets defined in drcInstructionsUniDrcV1() can refer to specific filters using their indices (see 6.4.11).

6.1.2.4 drcInstructionsBasic(), drcInstructionsUniDrc(), and drcInstructionsUniDrcV1()

A drcInstructions block includes information about one specific DRC set that can be applied to achieve a desired effect. This block can have the basic format or the uniDrc format. The basic format, drcInstructionsBasic(), contains a subset of information included in drcInstructionsUniDrc() that can be used to describe DRCs other than the ones specified in this document. The information included in drcInstructionsUniDrc() consists mainly of pre-defined description elements such as the DRC set effect and the DRC gain sequences that are applied. The *drcSetEffect* field contains several effect bits as listed in Table A.45. Multiple bits can be set unless otherwise noted. If no effect bit is set at all, the DRC set is ignored in the DRC set selection (see 6.3). Each drcInstructions block carries a unique non-zero identifier *drcSetId*. A *downmixId* is included to indicate if this DRC set applies to a certain downmix with this identifier. A *downmixId* of zero indicates that the DRC set is applied to the base layout. A *downmixId* of 0x7F indicates that the DRC set can be applied before or after the downmix. Since such a DRC can be applied to any downmix, it has only one channel group including all channels. If a “Ducking” bit is set in the *drcSetEffect* field, the DRC set is applied before any downmix specified by the downmix ID, i.e. the DRC set is always applied to the base layout and the downmix is generated thereafter. The *downmixId* 0x7F is not permitted for a ducking DRC set. In all other cases, the DRC set is applied to the channel configuration indicated by the *downmixId*.

The drcInstructionsUniDrcV1() payload is defined in Table 73. Compared to the drcInstructionsUniDrc() payload, it contains the same information plus several enhancements. However, the *downmixIDs* that appear in the two payloads are interpreted differently. For drcInstructionsUniDrcV1(), the *downmixID* indicates which downmix configuration is permitted in combination with this DRC, but it does not specify whether the DRC is applied to the downmix or the base layout. This is controlled by the *drcApplyToDownmix* flag instead.

The enhanced metadata of drcInstructionsUniDrcV1() compared to drcInstructionsUniDrc() includes references to target DRC characteristics and dynamic equalization filters. If a target characteristic is referenced, the corresponding DRC gain values shall be mapped to the target characteristic unless the host system overrides the target characteristic as specified in Table 17. If dynamic equalization filters are referenced, they shall be applied when the corresponding DRC set is selected and when this feature is supported by the decoder implementation (see 6.4.11).

A second DRC set may be specified for certain configurations. These configurations include cases where, for example, one DRC set is used for dynamic range compression and the other for clipping prevention ("Clipping" bit is set); or, for example, one DRC set is applied before and the other after the downmix. In those cases, the first DRC set contains a non-zero field *dependsOnDrcSet* that has the value of the *drcSetId* of the second DRC set it depends on. The declared DRC set effects of the first DRC set do not take into account the effects of the second DRC set. If the second DRC set is not designed to be used without combining it with another DRC set, the *noIndependentUse* flag shall be set to 1. In that case, the DRC set can only be used in combination with another DRC set as indicated by the *dependsOnDrcSet* field of the other set that is combined with it.

If a second DRC is specified by the *dependsOnDrcSet* field of a drcInstructionsUniDrcV1() payload, the combined DRCs cannot be applied if the DRC decoder downmix configuration does not match any *downmixID* in any of the two corresponding drcInstructionsUniDrcV1() payloads. These downmix configurations may include the output of the base layout without downmix, which is specified by a *downmixID*==0. Otherwise, the combined DRC is compatible with the downmix.

A DRC gain set is a set of DRC gain sequences, where the sequences are assigned to the bands of a multi-band DRC. For a single band DRC, the gain set contains just one gain sequence. Usually, each audio channel is assigned to a DRC gain set. A collection of channels assigned to the same DRC gain set is called "channel group". The assignment of a DRC gain set to a channel group is done in the order of first appearance of the set index when iterating through all channels (see also Table 15). A DRC gain set index *bsGainSetIndex*==0 indicates that the assigned channel will be passed through by the DRC tool without processing unless otherwise noted. Therefore, *bsGainSetIndex* is effectively 1-based, whereas the corresponding indices (*gainSetIndex*) are zero-based.

If subsequent channels are assigned the same gain set index, the field *repeatGainSetCount* indicates how many channels will have the same gain set not including the first.

The *drcLocation* field is used in the same way as the *drcLocation* field in the drcCoefficients (see 6.1.2.3). Certain entries of the *drcLocation* field allow adding drcInstructions information to gain sequences defined elsewhere. Some use cases are discussed in C.10.

The field *limiterPeakTarget* declares the peak target level used by the encoder-side DRC, if applicable. For example, if a limiter is used to generate the DRC gain sequence, it is configured to control the audio sample magnitude to not exceed this peak target level. *limiterPeakTarget* is represented in dBFS and encoded according to Table A.40.

If *limiterPeakTarget* is present, and the only *drcSetEffect* is "clipping prevention", the gain sequence is to be shifted by the negative sum of *loudnessNormalizationGainDb* and *limiterPeakTarget* if the negative sum is greater than 0. Afterwards, the gain sequence is saturated at the threshold of 0 dB so that only

negative gains (dB) occur. With this mechanism, it is possible to send gains for clipping prevention in expectation of a high *loudnessNormalizationGainDb*. If *loudnessNormalizationGainDb* is lower than expected, the gains are applied only as far as needed, and the dynamic range can be kept as high as possible.

If *gainScalingPresent* == 1, the gain scaling coefficients shall be applied to the channel group. If *gainOffsetPresent* == 1, the gain offset value shall be applied to the channel group as shown in Table 17. Similarly, if *duckingScalingPresent* == 1, the scaling factor shall be applied to the associated ducking gain sequence for that channel group.

The in-stream drcInstructions syntax is given in Table 70, Table 72 and Table 73. The syntax for the corresponding block for ISO/IEC 14496-12 is shown in Table 71 and Table 74. The corresponding blocks carry essentially the same information. Values that are identically included in both blocks are coded the same way except for *drcLocation*. Further information on the coding of *drcLocation* is defined in 6.1.2.3.

6.1.2.5 loudnessInfo() and loudnessInfoV1()

A loudnessInfo() block includes loudness and peak information. A downmix identifier and DRC set identifier indicate which configuration the information applies to. Hence, this block can be associated with the audio signal without DRC and without downmix, or with any specific DRC and/or downmix applied. If a DRC with a dependent DRC set is applied, the loudness information describes the output of the combined DRCs. A loudnessInfo() block can either represent an individual content item or the entire album. Typically, all content items of an album include identical album loudnessInfo() blocks.

If *downmixId* is zero, then loudnessInfo() applies to the base layout. If the *drcSetId* is zero, then loudnessInfo() applies to the audio signal without DRC processing. If *drcSetId* is 0x3F, then loudnessInfo() applies to any DRC processing including no DRC.

The fields *samplePeakLevel* and *truePeakLevel* represent the level of the maximum sample magnitude in dBFS and the True Peak in dBTP, respectively, of the associated audio content before or after audio encoding as defined in Reference [4]. The *measurementSystem* field includes standardized systems and others (see Table A.50). System 3 is defined as ITU-R BS.1770-4 with pre-processing. The pre-processing is a high-pass filter that models the typical limited frequency response of portable device loudspeakers. System 4 is defined as “User”. It means that the corresponding *methodValue* reflects a (subjective) user preference. System 5 is defined as “Expert/Panel”. It means that the corresponding *methodValue* represents a (subjective) expert or panel preference.

The *methodDefinition* field according to Table A.49 specifies how the *methodValue* is derived. The mixing level is compatible with “mixlevel” in ATSC A/52[1]. It indicates the absolute acoustic sound pressure level of an individual channel during the final audio mixing session. The peak mixing level is the acoustic level of a sine wave in a single channel whose peaks reach 100 percent in the PCM representation. The absolute SPL value is typically measured by means of pink noise with an RMS value of −20 or −30 dB with respect to the peak RMS sine wave level. The value of mixing level is not typically used within the DRC tool, but may be used by other parts of the audio reproduction system.

The room type field is compatible with “roomtyp” in ATSC A/52[1]. It indicates the type and calibration of the mixing room used for the final audio mixing session. The value of *roomtyp* is not typically used by the DRC tool, but may be used by other parts of the audio reproduction system.

The loudnessInfoSet() payload contains all loudnessInfo() blocks. The in-stream syntax of loudnessInfoSet() is given in Table 58. For the ISO base media file format, the slightly different syntax of “LoudnessBox” is used as defined in ISO/IEC 14496-12.

6.1.2.6 loudEqInstructions()

The loudEqInstructions() payload includes information relevant for the loudness equalization support. Each instance of this payload defines a set of loudness equalizer metadata and which combinations of downmix, DRC, and EQ it can be applied to. The metadata includes references to the corresponding DRC gain sequences and associated parameters needed to derive the acoustic level data. A typical way of generation and use of the metadata is given in D.2.10.

6.1.3 Derivation of peak and loudness values

The loudnessInfo() blocks provide optional values that describe loudness and peak. Several DRC decoder processes depend on these values, hence, when the loudness information is partially or entirely absent, fallback values are used as shown in Table 5. For peak values, a default value shall be used. Some other values can be drawn from the loudnessInfo() block of the base layout.

Table 5 — Default and fallback values of loudnessInfo

Value	Default	1st fallback: use value from loudnessInfo() of base layout with same DRCsetId	2nd fallback: use value from loudnessInfo() of the base layout without DRC
truePeakLevel	0.0	No	No
samplePeakLevel	0.0	No	No
programLoudness	Undefined	Yes	Yes
anchorLoudness	Undefined	Yes	Yes
loudnessRange	Undefined	No	No
Maximum loudness range	Undefined	No	No
Maximum momentary loudness	Undefined	No	No
Maximum short-term loudness	Undefined	No	No
Short-term loudness	Undefined	No	No
Mixing level	Undefined	Yes	Yes
Room type	Undefined	Yes	Yes

The applicable loudnessInfo() structure for determination of *programLoudness* and *anchorLoudness* is determined as specified in Table 6. The final loudness metadata value is extracted based on the selection steps specified in 6.10.2.

Table 6 — Determination of applicable loudnessInfo() structure for selection of *programLoudness* or *anchorLoudness* for a specific DRC set

```
getApplicableLoudnessInfoStructure(drcSetId, downmixIdRequested) {
  /* default value */
  loudnessInfo = getLoudnessInfoStructure(drcSetId, downmixIdRequested);
  /* fallback values */
  if (loudnessInfo == NULL) {
    loudnessInfo = getLoudnessInfoStructure(drcSetId, 0x7F);
  } else if (loudnessInfo == NULL) {
```

```
    loudnessInfo = getLoudnessInfoStructure(0x3F, downmixIdRequested);
  } else if (loudnessInfo == NULL) {
    loudnessInfo = getLoudnessInfoStructure(0, downmixIdRequested);
  } else if (loudnessInfo == NULL) {
    loudnessInfo = getLoudnessInfoStructure(0x3F, 0x7F);
  } else if (loudnessInfo == NULL) {
    loudnessInfo = getLoudnessInfoStructure(0, 0x7F);
  } else if (loudnessInfo == NULL) {
    loudnessInfo = getLoudnessInfoStructure(drcSetId, 0);
  } else if (loudnessInfo == NULL) {
    loudnessInfo = getLoudnessInfoStructure(0x3F, 0);
  } else if (loudnessInfo == NULL) {
    loudnessInfo = getLoudnessInfoStructure(0, 0);
  }
  return loudnessInfo;
}

getLoudnessInfoStructure(drcSetId, downmixId) {
  if (useAlbumMode == 1) count = loudnessInfoAlbumCount;
  else count = loudnessInfoCount;
  for (i=0; i<count; i++) {
    if (loudnessInfo[i]->drcSetId == drcSetId) && (loudnessInfo[i]->downmixId == downmixId) {
      for (j=0; j<loudnessInfo[i]->measurmentCount; j++) {
        if ((loudnessInfo[i]->measure[j]->methodDefinition==1)||(loudnessInfo[i]->measure[j]->methodDefinition==2)) {
          return loudnessInfo[i];
        }
      }
    }
  }
  return NULL;
}
```

The *signalPeakLevel* of a DRC set is determined as specified in Table 7, where peak related metadata entries are selected dependent on their availability and dependent on the *drcSetId*, and the requested *downmixId*. If no explicit peak information is available, *signalPeakLevel* is estimated from downmix coefficients and others. The estimates based on downmix coefficients hold for passive downmixers and might hold for specific active downmixers.

Table 7 — Determination of *signalPeakLevel* for a specific DRC set

```
getSignalPeakLevelForDrcSet (drcSetId, downmixIdRequested) {
  dmxId = downmixIdRequested;
  if truePeakLevelIsPresent(drcSetId, dmxId) {
    signalPeakLevel = getTruePeakLevel(drcSetId, dmxId);
  } else if samplePeakLevelIsPresent(drcSetId, dmxId) {
    signalPeakLevel = getSamplePeakLevel(drcSetId, dmxId);
  } else if truePeakLevelIsPresent(0x3F, dmxId) {
    signalPeakLevel = getTruePeakLevel(0x3F, dmxId);
  } else if samplePeakLevelIsPresent(0x3F, dmxId) {
    signalPeakLevel = getSamplePeakLevel(0x3F, dmxId);
```

```
    } else if limiterPeakTargetIsPresent(drcSetId, dmxId) {
        signalPeakLevel = getLimiterPeakTarget(drcSetId, dmxId);
    } else if (dmxId != 0) {
        signalPeakLevelTmp = 0.0;
        downmixPeakLevelLinear = 0.0;
        if downmixCoefficientsArePresent(dmxId) {
            for (i=0; i<targetChannelCount; i++) {
                downmixPeakLevelLinearTmp = 0.0;
                for (j=0; j<baseChannelCount; j++) {
                    downmixPeakLevelLinearTmp += pow(10.0, getDownmixCoefficient(dmxId, i, j)/20.0);
                }
                if (downmixPeakLevelLinear < downmixPeakLevelLinearTmp) {
                    downmixPeakLevelLinear = downmixPeakLevelLinearTmp;
                }
            }
        }
        if truePeakLevelIsPresent(drcSetId, 0) {
            signalPeakLevelTmp = getTruePeakLevel(drcSetId, 0);
        } else if samplePeakLevelIsPresent(drcSetId, 0) {
            signalPeakLevelTmp = getSamplePeakLevel(drcSetId, 0);
        } else if truePeakLevelIsPresent(0x3F, 0) {
            signalPeakLevelTmp = getTruePeakLevel(0x3F, 0);
        } else if samplePeakLevelIsPresent(0x3F, 0) {
            signalPeakLevelTmp = getSamplePeakLevel(0x3F, 0);
        } else if limiterPeakTargetIsPresent(drcSetId, 0) {
            signalPeakLevelTmp = getLimiterPeakTarget(drcSetId, 0);
        }
        signalPeakLevel = signalPeakLevelTmp + 20.0*log10(downmixPeakLevelLinear);
    } else {
        signalPeakLevel = 0.0;          /* worst case estimate */
    }
    return signalPeakLevel
}
```


Table 7 includes functions to check the availability and to retrieve peak-related information from loudnessInfo() and a drcInstructions block which can have the basic or uniDrc format. Table 8 shows pseudo code for some of the functions for the *truePeakLevel* and *limiterPeakTarget* based on downmixInstructions(), drcInstructionsUniDrc(), and loudnessInfo() payloads. The functions shall be adapted accordingly for the downmixInstructionsV1(), drcInstructionsUniDrcV1(), or loudnessInfoV1() payloads, if present. The functions for *samplePeakLevel* can be implemented by replacing *truePeakLevel* with *samplePeakLevel*.

Table 8 — Pseudo code for functions referenced in Table 7

```
truePeakLevelIsPresent(drcSetId, downmixId) {
    if (useAlbumMode == 1) count = loudnessInfoAlbumCount;
    else count = loudnessInfoCount;
    for (i=0; i<count; i++) {
        if (loudnessInfo[i]->drcSetId == drcSetId) && (loudnessInfo[i]->downmixId == downmixId) {
            if (loudnessInfo[i]->truePeakLevelPresent) return TRUE;
```

```
      }
   }
   return FALSE;
}

getTruePeakLevel(drcSetId, downmixId) {
   if (useAlbumMode == 1) count = loudnessInfoAlbumCount;
   else count = loudnessInfoCount;
   for (i=0; i<count; i++) {
      if (loudnessInfo[i]->drcSetId == drcSetId) && (loudnessInfo[i]->downmixId == downmixId) {
         if (loudnessInfo[i]->truePeakLevelPresent) {
            return (loudnessInfo[i]->truePeakLevel);
         }
      }
   }
   return error;
}

limiterPeakTargetIsPresent(drcSetId, downmixId) {
   for (i=0; i<drcInstructionsCount; i++) {
      if    (drcInstructions[i]->drcSetId == drcSetId) &&
            ((drcInstructions[i]->downmixId == downmixId) ||
            (drcInstructions[i]->downmixId == 0x7F)) {
         if (drcInstructions[i]->limiterPeakTargetPresent) return TRUE;
      }
   }
   return FALSE;
}

getlimiterPeakTarget(drcSetId, downmixId) {
   for (i=0; i<drcInstructionsCount; i++) {
      if    (drcInstructions[i]->drcSetId == drcSetId) &&
            ((drcInstructions[i]->downmixId == downmixId) ||
            (drcInstructions[i]->downmixId == 0x7F)) {
         if (drcInstructions[i]->limiterPeakTargetPresent)) {
            return (drcInstructions[i]->limiterPeakTarget);
         }
      }
   }
   return error;
}
downmixCoefficientsArePresent(downmixId) {
   for (i=0; i<downmixInstructionsCount; i++) {
      if (downmixInstructions[i]->downmixId == downmixId) {
         if (downmixInstructions[i]->downmixCoefficientsPresent) return TRUE;
      }
   }
   return FALSE;
}
```

```
getDownmixCoefficient(downmixId, outChan, inChan) {
   for (i=0; i<downmixInstructionsCount; i++) {
      if (downmixInstructions[i]->downmixId == downmixId) {
         if (downmixInstructions[i]->downmixCoefficientsPresent) {
            return (downmixInstructions[i]->downmixCoefficient[outChan][inChan]);
         }
      }
   }
   return error;
}
```

6.2 Dynamic DRC gain payload

The dynamic gain sequences for all DRCs are received in-stream or via a metadata track using the uniDrcGain() syntax given in Table 55. Each access unit contains gain sequences for the duration of *drcFrameSize* samples that are decoded according to 6.4.5.

The last part of the uniDrcGain() can include future extension payloads. If a *uniDrcGainExtType* is received that is different from UNIDRCGAINEXT_TERM, the extension payload (*otherBit*) shall be read and discarded.

When gain values are stored in a separate meta-data track under the sample entry code "unid" as described in 6.1.2.3, each sample is a uniDrcGain() padded with 0 to 7 trailing zero bits to the next byte boundary.

6.3 DRC set selection

6.3.1 Overview

A bitstream can carry multiple DRCs for various purposes. At the decoder, the DRC set that best matches the requirements for the given playback scenario is selected. The selection process is performed in three stages:

1. A pre-selection that discards all DRC sets that are not applicable because they do not match a target channel configuration, do not support the decoder target loudness, or have more clipping than requested (see 6.3.2).

2. A main selection process based on requested DRC set features (see 6.3.3).

3. A final selection that picks a single DRC set if multiple DRC sets are applicable (see 6.3.4).

The most relevant metadata for the selection process is summarized in Table 9. The bit fields of the *drcSetEffect* field are described in detail in Table A.45. All parameters that can be supplied by the host to control loudness normalization and dynamic range compression are summarized in Table A.102 and Table A.103, respectively.

If the DRC tool is configured to support equalization (EQ) according to 6.8, the selection process includes the EQ-related metadata and the final selection also considers requests for specific EQ, such as EQ dependent on the playback room size. Otherwise, the preselection discards all DRC sets that require EQ, i.e. which have a value of *requiresEq*==1.

DRC sets with only a “Fade” or “Ducking” effect are automatically selected by the decoder without using the three-stage selection process. DRC sets with other features can be requested by using DRC decoder settings as described below.

The pool of DRC sets that is subject to the three-stage selection process comprises not only the DRC sets defined in the bitstream (except for “Fade” and “Ducking”) but also virtual DRC sets generated in the DRC tool. The virtual DRC sets are placeholders for the cases where no compression is applied to the audio signal, hence their drcSetEffect bits are zero and they correspond to the DRC effect request “None”.

Since the DRC set selection is driven by requests, DRC processing shall not be applied if it is not requested. More specifically, dynamic range compression shall not be applied if there is no corresponding request, i.e. *dynamicRangeControlInterfacePresent* == 0 (see Annex B); loudness normalization shall not be applied if it is not requested, i.e. *loudnessNormalizationInterfacePresent* == 0 (see Annex B).

Conceptually it is possible to integrate other DRC sets specified by the drcInstructionsBasic() and drcCoefficientsBasic() syntax in the selection process. However, this is out of scope and not part of this document.

Table 9 — Most relevant metadata for DRC selection at the decoder

Field	Type
drcInstructionsUniDrc->downmixId	Identifier
drcInstructionsUniDrc->limiterPeakTarget	Value
drcInstructionsUniDrc->drcSetEffect	Bit field
drcInstructionsUniDrc->noIndependentUse	Flag
drcInstructionsUniDrc->drcSetTargetLoudnessValueUpper/-Lower	Value
loudnessInfo->truePeakLevel	Value
loudnessInfo->samplePeakLevel	Value
loudnessInfo->program loudness	Value
loudnessInfo->anchor loudness	Value
loudnessInfo->maximum short-term loudness	Value
loudnessInfo->maximum momentary loudness	Value
loudnessInfo->maximum loudness range	Value
drcCoefficientsUniDrc->drcCharacteristic[sequence][band]	Index
drcCoefficientsUniDrc->bandCount[sequence]	Value

6.3.2 Pre-selection based on Signal Properties and Decoder Configuration

6.3.2.1 Overview

The pre-selection selects all DRC sets that fulfill all requirements listed in Table 10. All available DRC sets are analysed in the given order of steps. If no DRC set is selected, no DRC can be applied except for fading or ducking.

Table 10 — Requirements for DRC pre-selection

#	Requirement	Applicability	Comment
1	*downmixId* of DRC set matches the requested *downmixId*.	If a *downmixId* is requested.	See 6.3.2.2.1.

#	Requirement	Applicability	Comment
2	Output channel layout of DRC set matches the requested layout.	If a target channel layout is requested.	See 6.3.2.2.1.
3	Channel count of DRC set matches the requested channel count.	If a target channel count is requested.	See 6.3.2.2.1.
4	The DRC set is not a “Fade-” or “Ducking-” only DRC set.	Always.	DRC sets with “Fade” or “Ducking” effect are selected automatically. They are not subject to this selection process.
5	The number of DRC bands is supported.	Always.	DRC sets that exceed the number of supported DRC bands are discarded. For time-domain DRC, the maximum is four bands.
6	Independent use of DRC set is permitted.	If the DRC set is not used in combination with another DRC set.	DRC sets with a *noIndependentUse* flag value of 1 can only be used in combination with a second DRC set.
7	DRC sets that require EQ are only permitted if EQ is supported.	For all drcInstructionsUniDrcV1() payloads.	If EQ is not supported, DRC sets with *requiresEq*==1 are discarded.
8	The range of the target loudness specified for a DRC set has to include the requested decoder target loudness.	If *drcSetTargetLoudnessPresent*==1 and no explicit peak information is available for that DRC set.	See 6.3.2.2.2.
9	Clipping is minimized.	Except for DRC sets which were already selected in pre-selection step #8.	See 6.3.2.2.3.
NOTE Pre-selection steps #8 and #9 are interpreted as pre-selection steps #7 and #8 in the first edition of this document (ISO/IEC 23003-4:2015). Pre-selection step #7 related to EQ support is first available with the second edition of this document (ISO/IEC 23003-4:2019).			

6.3.2.2 Detailed description of pre-selection steps

6.3.2.2.1 Pre-selection based on downmix ID, channel layout, or channel count (#1,2,3)

Requests for downmix IDs, target channel layout, or target channel count are supported. Only one of these requests will be used based on the following priority:

1. downmix ID(s)
2. target channel layout
3. target channel count

If no downmix ID is requested, the request(s) will be mapped to one or more matching *downmixIds* that are used in the pre-selection process as defined in the following.

If a target channel layout is requested but a downmix ID is not requested, the channel layout is mapped to downmix IDs with a matching layout. If only a target channel count is requested, it is mapped to downmix IDs with a matching target channel count. If no matching downmix IDs can be found, no DRC set can be applied except for fading or ducking.

If no request is present, a *downmixId* of 0x0 (base layout) will be used for the pre-selection.

A DRC set is selected if the requested downmix ID (or the downmix ID, which was generated from a different kind of request) matches one of the defined *downmixId* values of the DRC set. DRC sets which define a *downmixId* of 0x0 or 0x7F automatically pass the pre-selection #1,2,3.

A downmix ID list contains all downmix IDs to be used for pre-selection as described in the previous steps. The pre-selection for these requests is done by selecting all DRC sets that have one of the *downmixIds* of the list or a *downmixId* of 0x0 or 0x7F. A requested *downmixId* of 0x7F is meaningless and therefore not permitted.

6.3.2.2.2 Pre-selection based on drcSetTargetLoudness (#8)

This pre-selection step addresses only DRC sets for which *drcSetTargetLoudnessPresent* equals one and for which no explicit peak information is available. From the DRC sets which match this criterion, only those are selected whose range defined by *drcTargetLoudnessValueUpper/-Lower* includes the requested decoder target loudness (*targetLoudness*). A range check shall include the upper boundary value and exclude the lower boundary value. Pre-selection step #9 is omitted for DRC sets selected in this step. Explicit peak information for a specific DRC set is available if at least one of the following results is TRUE:

— truePeakLevelIsPresent(*drcSetId*, *downmixIdRequested*)

— samplePeakLevelIsPresent(*drcSetId*, *downmixIdRequested*)

— limiterPeakTargetIsPresent(*drcSetId*, *downmixIdOfDrcSet*)

6.3.2.2.3 Pre-selection based on output peak level (#9)

This pre-selection step addresses the problem that can arise if due to loudness normalization or downmixing the maximum peak value exceeds full scale. Two mechanisms can be used to avoid clipping, a DRC set that reduces or eliminates clipping and/or a peak limiter. DRC sets resulting in too high peak values at the output possibly need to be discarded. The peak value of the output signal is computed as

outputPeakLevel = *signalPeakLevel* + *loudnessNormalizationGainDb*

where

signalPeakLevel	is the peak value of the signal in dBFS according to Table 7;
loudnessNormalizationGainDb	is the scaling factor for the loudness normalization in dB.

NOTE The *loudnessNormalizationGainDb* can be specific for each DRC set if individual loudnessInfo() blocks are present for each DRC set.

All DRC sets are discarded whose *outputPeakLevel* exceeds *outputPeakLevelMax.* The recommended value of *outputPeakLevelMax* is 0 dBFS when no peak limiter is applied after the DRC tool. If a peak

limiter is subsequently applied, it is recommended to set *outputPeakLevelMax* to the maximum peak value the peak limiter can handle without introducing severe and audible distortions. By default, the DRC tool assumes that no peak limiter will be applied. The default value for *outputPeakLevelMax* is 0 dBFS. If a peak limiter is applied (*peakLimiterPresent*==1), the default value for *outputPeakLevelMax* is 6 dBFS.

If no DRC set was selected so far (including the pre-selection in step #8), select the DRC sets for which *drcSetTargetLoudnessPresent* equals one and whose range defined by *drcTargetLoudnessValueUpper/-Lower* includes the requested decoder target loudness. If after that no DRC set was selected, the requirement of clipping prevention needs to be relaxed until at least one DRC set is selected. Therefore, the DRC sets with the lowest *outputPeakLevel* are selected. In addition, the DRC sets are selected whose *outputPeakLevel* does not exceed the lowest *outputPeakLevel* by more than 1.0 dB.

At this point, the output signal can have audible distortions, either due to clipping or because a peak limiter is applied at too high levels. This is mitigated by lowering the output level resulting in an intentional deviation from the target loudness. The parameter *loudnessDeviationMax* (which is 63 dB by default) limits the gain reduction that is applied to reduce the distortions (e.g. *loudnessDeviationMax* of 3 dB permits a gain adjustment of −3 dB).

6.3.3 Selection based on requests

6.3.3.1 Overview

In the following, the selected DRC sets are matched with requested features that can describe certain aspects of a DRC set. The features are organized in an ordered list that is given to the DRC decoder. One or more features can be requested (see also Annex E). A feature can be requested multiple times. The decoder will search the available DRC sets to find the best match according to the following rules. Table 11 lists DRC features that are supported for the DRC search.

Table 11 — Requestable features of DRC sets

Index	DRC feature	Comment
0	“Effect type”	Specifies the type of DRC effect
1	“Dynamic range”	Specifies the dynamic range, see Table 13
2	“DRC characteristic”	Specifies the DRC characteristic at the encoder

The decoder works through the feature list item-by-item starting from the first. Conceptually, it takes for each item the selected DRC sets and selects only those that match the requested feature. The following specifies the selection rules for each requested feature in detail. If no features are requested, the selection process is performed as if the effect type feature with effect type “None” has been requested as described in the following section. If that request does not succeed, another feature request attempt is issued using the “General” effect type with the fallback effect types “Night”, “Noisy”, “Limited”, “LowLevel”.

6.3.3.2 Sub-selection by requesting an effect type feature

The effect type as specified by the field *drcSetEffect* in Table A.45 describes the result of applying the DRC set. For DRC sets that carry a non-zero entry in the *dependsOnDrcSet* field, the valid effect types for the combination of both sets are obtained from the effect types of both sets (the depending DRC set and the one with a corresponding *dependsOnDrcSet* value).

Table 12 lists all effect types that can be requested. One effect type can be requested at a time. Requests for effect types can be repeated multiple times with different effect types. From the DRC sets selected so far, all DRC sets that match the requested effect type are selected.

An effect type request is ignored if none of the selected DRC sets match. Consequently, the sub-selection process proceeds with the results of the previous request, if applicable; otherwise the results of the pre-selection are used.

The list of requested effect types contains the desired effect types. Additionally, fallback effect types can follow the desired effect types. The fallback effect types are specified to allow selecting a DRC set even if DRC sets with the desired effect types are not available.

After a request sequence of desired effect types, the sub-selection ends if one or more DRC sets are selected and any fallback requests are ignored. Otherwise, the sub-selection continues with the fallback effects. In that case, the sub-selection ends if one or more matching DRC sets have been selected during the processing of fallback requests and the remaining fallback requests are ignored.

The requests can be specified as a list of requested effect types together with an index of the first fallback request in the list. If, for example, only one single desired effect type is specified, the fallback requests start at position 2. More detailed explanations are given in Annex E.

Table 12 — Requestable DRC effect types and short names for reference

Index	drcSetEffect	Short name	Description
0	None	“None”	A DRC set that has no dynamic compression, for instance “Clipping”, or applying no DRC set for compression.
1	Late night	“Night”	For quiet environment, listening at low level, avoiding to disturb others.
2	Noisy environment	“Noisy”	Optimized to get the best experience in noisy environments, for instance by amplifying soft sections.
3	Limited playback range	“Limited”	Reduced dynamic range to improve quality on playback devices with limited dynamic range.
4	Low playback level	“LowLevel”	Listening at a low playback level.
5	Dialog enhancement	“Dialog”	The main effect is a more prominent dialogue within the content.
6	General compression	“General”	A DRC effect that reduces the dynamic range and is applicable to multiple playback scenarios.
7	Dynamic expansion	“Expand”	Dynamics enhancement.
8	Artistic effect	“Artistic”	To create an artistic sound effect.

6.3.3.3 Sub-selection by requesting a “Dynamic Range” value

The decoder can receive a request for a single Dynamic Range measurement value or value range. The measurement is based on one of those provided in Table 13. If a value is specified, all DRC sets are selected with the closest matching value of the feature. If a range is specified, all DRC sets are selected whose feature value is within that range.

Each of the measurement values for “short-term loudness”, “momentary loudness”, and “top of loudness range” can be given for multiple measurement systems. Permitted measurement systems include EBU R128 and BS.1771-1. The first system shall be selected when searching the available values in order of

increasing measurement system index (see Table A.50) including all reserved indices for future updates, i.e. if a reserved index is found, it shall be interpreted as a valid measurement system.

The program loudness values used in Table 13 shall be based on the following measurement systems (see Table A.50):

1. RMS_C
2. RMS_B
3. RMS_A
4. BS.1770-4

If multiple values are given, choose the first available using the order of the list.

Table 13 — Requestable dynamic range measurement values

Index	Label	Dynamic range measurement	Comment
0	StA	"Short-term loudness peak to average"	Max. short-term loudness minus program loudness (see 6.10.2)
1	MtA	"Momentary loudness peak to average"	Max. momentary loudness minus program loudness (see 6.10.2)
2	TtA	"Top of loudness range to average"	Top of loudness range minus program loudness (see 6.10.2)

6.3.3.4 Sub-selection by requesting a "DRC characteristic"

From the selected DRC sets, all sets are selected with the closest DRC characteristic index according to a matching order. If different DRC characteristics are used within the same DRC set, only the DRC characteristic indices are taken that are in the matching order list. The remaining ones are ignored. If no DRC set is found for the matching order of the requested DRC characteristic in Table 14, this selection step is ignored.

Table 14 — Matching order for DRC characteristic

		Requested DRC characteristic											
		1	2	3	4	5	6	7	8	9	10	11	c; c > 11
Matching order	first	1	2	3	4	5	6	7	8	9	10	11	c
	second	2	3	4	5	6	5	9	10	7	8	10	-
	third	-	1	2	3	4	-	-	-	-	-	9	-

6.3.4 Final selection

This clause uses the term "multiple DRC sets" for DRC sets that are independent of each other and do not include ducking or fading effects.

If the DRC tool is configured to support EQ and if there are still multiple DRC sets selected, select the ones that can be combined with an EQ of the requested "EQ purpose" (see B.3.7.2). If no DRC set can be selected, select the ones that can be combined with an EQ of the default "EQ purpose". If still no DRC set can be selected, ignore this step.

If there are still multiple DRC sets selected, select the ones that result in an output peak value of 0.0 or less. If no such DRC set is found, from the ones selected, select the DRC sets that exceed the threshold by the minimum amount. DRC sets selected in pre-selection step #8 are assumed to have an output peak value of 0.0 or less.

If there are still multiple DRC sets selected, select the ones that exactly match the requested downmixId.

If there are still multiple DRC sets selected, select the ones with the minimum number of effect types where the effect bit for "General" is ignored. The reason for ignoring the effect bit "General" is that otherwise DRC sets without "General" would be unjustifiably preferred.

If there are still multiple DRC sets selected, discard any DRC set that was selected in the pre-selection step dealing with *drcSetTargetLoudness*. If all selected DRC sets are discarded in this step, select the DRC set with the smallest *drcSetTargetLoudnessValueUpper* instead.

If there are still multiple DRC sets selected, select the DRC sets for which *drcSetTargetLoudnessPresent* equals one and whose range defined by *drcSetTargetLoudnessValueUpper/-Lower* includes the requested decoder target loudness. If multiple DRC sets are selected, select the DRC set with the smallest *drcSetTargetLoudnessValueUpper*.

If there are still multiple DRC sets selected, select the DRC sets with the largest peak value. The output peak value also depends on the modifications of the gains as described in 6.4.6.

If there are still multiple DRC sets selected, choose the one with the largest *drcSetId*.

6.3.5 Applying multiple DRC sets

In the following cases, multiple DRC sets are applied simultaneously. First, if the DRC set selected in 6.3.4 carries a non-zero entry in the *dependsOnDrcSet* field, the depending DRC set is applied together with the selected one. Second, if a DRC set with "Fade" or "Ducking" effect was automatically selected, it is applied simultaneously with the DRC set selected in 6.3.4. Thus, if the DRC set selected in 6.3.4 has a non-zero *dependsOnDrcSet* value, a total of three DRC sets are applied, which is the maximum number permitted. If all three DRC sets are applied to the same layout (*downmixId*), the DRC set referenced in the *dependsOnDrcSet* field shall be applied first, the DRC set selected in 6.3.4 shall be applied thereafter, and the DRC set with "Fade" or "Ducking" effect shall be applied last. If none of the applied DRC sets is a parametric DRC (see 6.6), the DRC set with "Fade" or "Ducking" can alternatively be processed first. If only two DRC sets are applied to the same layout (*downmixId*), the same order applies. If a DRC set with "Fade" effect and another DRC set with "Ducking" effect were both automatically selected, the DRC set with "Fade" effect is ignored. A DRC set with "Fade" effect shall be applied after the downmix, if present, if any of the applied DRC sets is a parametric DRC.

6.3.6 Album mode

When the playback system is in album mode, i.e. successive songs of an album are played back, the DRC selection should be applied first using all DRCs in the loudnessInfo blocks for albums. These blocks may contain *drcSetIds* that point to matching DRCs. Only if there is no match with any DRC in a loudnessInfo block for albums, the selection should proceed to apply the logic described in 6.3.2 to all available DRCs.

In album mode, any "Fade" DRC is not applied. If not in album mode, if an applicable "Fade" DRC exists, it shall be applied. The "Fade" DRC can be applied simultaneously with any other DRC except "Ducking".

6.3.7 Ducking

The base layout and each specific downmix with a unique *downmixId* can have a maximum of one DRC set with a ducking effect. During configuration, the decoder scans all available DRC sets for the active downmix to identify the applicable DRC set for ducking if present. If ducking DRC sets are both defined

for the base layout and the active downmix, select the one that exactly matches the active downmix. If a ducking DRC set is identified and the associated overlaid audio signal is active, the ducking gain sequence is automatically applied to all channels except those that are members of the channel group associated with the ducking DRC set (*drcSetEffect*=="Duck other") or alternatively to all channels that are members of the channel group associated with the ducking DRC set (*drcSetEffect*=="Duck self"). The overlaid audio is defined to be active if at least one non-zero downmix coefficient is applied to it.

Ducking is always applied before any downmix, i.e to the base layout. Hence, the DRC channel groups for the ducking process refer to the base layout. The *downmixId* of the corresponding drcInstructionsUniDrc() indicates how to generate the downmix after the ducking was applied.

A ducking DRC set with *downmixId* 0x0 (baseLayout) is automatically applied independent of the requested *downmixId*. It is therefore recommended to define ducking DRC sets with *downmixId* 0x0 only for specific use cases, where the ducking DRC set should be always applied when DRC processing is enabled.

6.3.8 Precedence

If a bitstream contains the described DRC metadata and other DRC metadata such as MPEG light or heavy compression, the described metadata will take precedence unless the decoder is instructed to apply the other DRC metadata.

6.4 Time domain DRC application

6.4.1 Overview

The DRC gain can be applied to the time-domain audio signal or to the sub-band signals of the audio decoder. The following text first includes a full description of the time-domain DRC. Subsequently, the necessary modifications of the time-domain DRC are described to achieve DRC in sub-bands.

Since the encoder provides only sparsely sampled gain values, the decoder applies interpolation to achieve a smooth gain transition between the samples (see also Annex D for DRC gain generation and encoding). The sample rate of the interpolated gain is the audio sample rate. Each gain sequence can be configured to use either linear interpolation or spline interpolation. For linear interpolation, the interpolated values of one segment between two subsequent gain samples (nodes) are derived from the two gain samples at both ends of the segment. For spline interpolation, they are additionally derived from their slope (derivative). Hence, when transitioning from one segment to the next, the first derivative is continuous as both segments have the same slope at the transition point.

For applications of the DRC tool in tandem with an audio codec, the following parameters are provided to adjust the DRC frame size and time resolution so that codec and DRC processing can be done most efficiently in terms of complexity and delay. The parameters are:

— DRC frame size in units of the audio sample interval

— *deltaTmin* in units of the audio sample interval

— delay mode.

These parameters have default values, but an audio codec specification may override the defaults.

6.4.2 Framing

The DRC gain information is organized in DRC frames. Each DRC frame contains DRC data to generate the DRC gain for the duration of a DRC frame. The DRC frame duration is constant for a given audio item and it is a multiple of the audio sample interval. DRC frames do not overlap.

In practice, whenever suitable, the DRC frame size M_{DRC} is recommended to correspond to the same duration as the codec's frame size to minimize delay and complexity. This is the default setting for frame sizes of 1 ms and more. For audio formats with a frame size below 1 ms, as is typically the case for PCM, the default DRC frame size is 32 *deltaTmin* as defined in Formula (1). The default frame size is used, unless the frame size is specified by *drcFrameSize* in the bitstream.

6.4.3 Time resolution

The DRC tool uses a uniform time grid to generate a sparse representation of the DRC gain. The spacing of this grid defines the highest available time resolution *deltaTmin*. The unit of *deltaTmin* is one sample interval at the audio sample rate. For efficiency, *deltaTmin* is chosen to be an integer multiple of the audio sample interval with a corresponding duration between 0.5 ms and 1.0 ms. Preferably, *deltaTmin* is an integer power of 2, so that sample rates can be efficiently converted between audio and DRC. The default values are computed based on Formulae (1) and (2):

$$f_s 0.0005 \text{ s} < deltaTmin \leq f_s 0.001 \text{ s} \tag{1}$$

and

$$deltaTmin = 2^M \tag{2}$$

where

f_s is the audio sample rate, in Hz;

M is the non-negative integer exponent.

Audio sample rates smaller than 1 kHz are not permitted.

For certain DRC frame sizes, it is not possible to find a value of M that fulfils Formula (2) and which results in a *deltaTmin* value that is an integer factor of the DRC frame size. In that case, the default value of *deltaTmin* is the integer value that results in a sample interval with the smallest deviation from 0.75 ms and which is an integer factor of the DRC frame size. The deviation ε is defined in Formula (3):

$$\varepsilon = \left| 1 - \frac{\text{deltaTmin}}{f_s 0.00075s} \right| \tag{3}$$

If there are several values of deltaTmin that result in the same deviation, the largest value shall be chosen.

Alternatively, the value of *deltaTmin* can be transmitted in the DRC bitstream field *timeDeltaMin*. If it is present, it overrides the default value.

6.4.4 Time alignment

The time alignment specifies the temporal location of each gain sample within a block of *deltaTmin* audio sample intervals. The alignment is conveyed by the *timeAlignment* field in drcCoefficientsUniDrc(). A value of 0 indicates that the gain sample is located at the last audio sample interval of the block, i.e. the sample index at the audio rate is *deltaTmin*-1 assuming that index 0 is the first audio sample interval in the block. A value of 1 indicates that the gain value is located in the centre of the block, i.e. at the sample index of floor((*deltaTmin*-1)/2).

In low-delay mode, a *timeAlignment* value of 1 is not supported. Hence, *timeAlignment* is set to 0 in low-delay mode. The *timeAlignment* value for the gain sequences of all *channelGroups* within a DRC set shall be identical.

6.4.5 Decoding

The DRC gains are derived from linear interpolation or spline interpolation, which is determined by the node coordinates and, in case of spline interpolation, by their associated slopes. The decoding process of the node coordinates consists of the following sequence of tasks:

1) Gather the DRC configuration information (once);

2) Select a DRC set (once);

3) Parse the DRC gain bitstream (per DRC frame);

4) Apply the code tables including Huffman decoding to decode the quantized values (per DRC frame);

5) Undo the differential encoding (per DRC frame).

The basic in-stream DRC configuration information is fully specified in this document including some parts of ISO base media file format that are not included in ISO/IEC 14496-12. The configuration of the core DRC gain decoding is essentially the same for both cases. The following configuration parameters are most relevant for decoding:

— The number of gain sequences: *nDrcGainSequences*;

— The profile for DRC gain coding: *drcGainCodingProfile*;

— The type of DRC gain interpolation: *gainInterpolationType*;

— The assignment of a gain set to each channel. Channels using the same gain set are referred to as channel groups. The total number of groups is *nDrcChannelGroups* (see Table 15);

— The number of DRC bands in a group: *nDrcBands*.

Given these parameters, the part of the DRC bitstream containing the DRC gains (uniDrcGain()) can be parsed and decoded using the algorithms of Table 16 up to Table 22 and the codes of Table A.1 and the following ones based on *gainCodingProfile*.

In the ISO base media file format, the audio content may be carried in multiple tracks where a base track contains the DRC metadata for all tracks. The additional tracks are referenced by the base track using a track reference of type "adda" (additional audio). The channels are all the channels in the base track, plus all the channels in track(s) referenced, in the order of the references. The channel groups apply to all those channels (even if they are channels in a track that is disabled or not currently being played).

Table 15 — Derivation of drcChannelGroups from gainSetIndices

```
uniqueIndex = {-10, -10, ...};
k=0;
if ((drcSetEffect & (3<<10)) != 0) {            /* Ducking */
    uniqueScaling = {-10, -10, ...};
    for (i=0; i<channelCount; i++) {
        match = FALSE;
```



```
            idx = gainSetIndex[i];
            factor = duckingScaling[i];
            if (idx<0) channelGroupForChannel[i] = -1;
            else {
                for (n=0; n<k; n++) {
                    if ((uniqueIndex[n] == idx) && (uniqueScaling[n] == factor)) {
                        match = TRUE;
                        channelGroupForChannel[i] = n;
                        break;
                    }
                }
                if (match == FALSE) {
                    uniqueIndex[k] = idx;
                    uniqueScaling[k] = factor;
                    channelGroupForChannel[i] = k;
                    k++;
                }
            }
        }
    }
    else {
        for (i=0; i<channelCount; i++) {
            match = FALSE;
            idx = gainSetIndex[i];
            if (idx<0) channelGroupForChannel[i] = -1;
            else {
                for (n=0; n<k; n++) {
                    if (uniqueIndex[n] == idx) {
                        match = TRUE;
                        channelGroupForChannel[i] = n;
                        break;
                    }
                }
                if (match == FALSE) {
                    uniqueIndex[k] = idx;
                    channelGroupForChannel[i] = k;
                    k++;
                }
            }
        }
    }
    nDrcChannelGroups = k;
```


The application of codes is expressed in Table 16 by the pseudo-functions decodeInitialGain(), decodeDeltaGain(), decodeTimeDelta(), and decodeSlope(). Differentially encoded values are then converted into absolute values according to Table 16. The decoded result is represented by the gain values *gDRC*[*g*][b][*k*], the time values *tDRC*[*g*][b][*k*], and the slope values *sDRC*[*g*][b][*k*] where *g* is the channel group index, *b* is the band index, and *k* is the node index. The DRC gain sequences are assigned to channel groups and DRC bands using the information in the drcCoefficientsUniDrc() and drcInstructionsUniDrc() payloads or their V1 versions. For linear interpolation, the slope values are set

to 0. They will be neglected during interpolation. Time values are integer numbers relative to the beginning of the DRC frame in units of *deltaTmin*. The audio sample that coincides with the beginning of the DRC frame has a time value of *tDRC*=0.

For *gainCodingProfile* == 3, no dynamic gain sequence is transmitted. Hence, the decoded values are constant. This is useful for generating a constant DRC gain based on the *gainOffset* value.

Table 16 — Decoding of DRC gain sample coordinates and slopes in the dB domain

```
if (timeAlignment==0) {
    timeOffset = -1;
}
else {
    timeOffset = - deltaTmin + floor((deltaTmin-1)/2);
}
for(g=0; g<nDrcChannelGroups; g++) {
    if (gainCodingProfile[g]==3) {
        nDrcBands[g] = 1;
        nNodes[g][b] = 1;
        gDRC[g][b][0] = 0.0;
        tDRC[g][b][0] = drcFrameSize + timeOffset;
        sDRC[g][b][0] = 0.0;
    }
    else {
        for(b=0; b<nDrcBands[g]; b++) {
            gDRC[g][b][0] = decodeInitialGain(gainInitialCode[g][b]);
            if (drcGainCodingMode[g][b] == 0) {
                /* "simple" mode */
                tDRC[g][b][0] = drcFrameSize + timeOffset;
                sDRC[g][b][0] = 0.0;
            }
            else {
                for (k=1; k<nNodes[g][b]; k++) {
                    gDRC[g][b][k] = gDRC[g][b][k-1]+decodeDeltaGain(gainDeltaCode[g][b][k-1]);
                }
                tmp_timeOffset = timeOffset;
                if (frameEndFlag[g][b] == 1) {
                    nodeResFlag = 0;
                    for (k=0; k<nNodes[g][b]-1; k++) {
                        tDRC_Buf = tmp_timeOffset + deltaTmin * decodeTimeDelta(timeDeltaCode[g][b][k]);
                        if (tDRC_Buf > drcFrameSize + timeOffset) {          /* nodes from node reservoir */
                            if (nodeResFlag == 0) {
                                tDRC[g][b][k] = drcFrameSize + timeOffset;
                                nodeResFlag = 1;
                            }
                            tDRC[g][b][k+1] = tDRC_Buf;
                        }
                        else {
                           tDRC[g][b][k] = tDRC_Buf;
                        }
                        tmp_timeOffset = tDRC_Buf;
```

```
                    }
                    if (nodeResFlag == 0) {
                        tDRC[g][b][k] = drcFrameSize + timeOffset;
                    }
                }
                else {
                    for (k=0; k<nNodes[g][b]; k++) {
                        tDRC[g][b][k] = tmp_timeOffset + deltaTmin * decodeTimeDelta(timeDeltaCode[g][b][k]);
                        tmp_timeOffset = tDrc[g][b][k];
                    }
                }
                for (k=0; k<nNodes[g][b]; k++) {
                    if (gainInterpolationType == 0) {
                        sDRC[g][b][k] = decodeSlope (slopeCode[g][b][k]);
                    }
                    else {
                        sDrc[g][b][k] = 0.0; /* slope will be neglected */
                    }
                }
            }
        }
    }
}
```

6.4.6 Gain modifications and interpolation

Before the gain can be applied to the audio signal, it shall be converted to the linear domain and gain values between gain samples shall be interpolated. To achieve lower complexity, the dB to linear conversion is done before the interpolation. Hence, the interpolation process is entirely done in the linear domain. Both the gain modification and conversion to the linear domain are done using the pseudo code of Table 17. The input variables are the gain sample and slope in the dB domain. The output consists of the gain sample and slope in the linear domain.

As described in E.2.4 and E.4, there are several ways to adapt the DRC characteristics in the DRC tool decoder. These adjustments are applied to the decoded gain samples in the dB domain. The function toLinear() includes all necessary steps to generate a linear gain sample from the logarithmic value in dB (see Table 17). It contains a mapping function mapGain() (see Table 18) that supports modifications of the DRC gain values with the purpose of achieving a different compression characteristic than the one used in the encoder. The mapping can be controlled by the host (*drcCharacteristicTarget* > 0) which selects one of the target characteristics based on an index. Otherwise, if specified in the drcInstructionsUniDrcV1() payload, a target characteristic is used and the DRC gain values are mapped to it using the mapGain() function. In that case, different characteristics or scalings can be applied independently to positive and negative gains (see Table 17). Otherwise, the encoder characteristic will not be replaced. A modified characteristic can be generated based on the encoder compression characteristic that is conveyed in the DRC configuration. Moreover, a compression and boost factor is supported to scale negative and positive gains, respectively. These factors have a value of 1.0, unless values in the range [0,1] are supplied by the user. Similarly, an encoder-controlled scaling is applied when *gainScalingPresent*==1 using the scale factors *attenuationScaling* and *amplificationScaling*. When ducking is active, the ducking gains in dB are scaled by the factor *duckingScaling,* if present. The *duckingScaling* factors are conveyed in the drcInstructionsUniDrc() payload for the channel they are applied to, which is in contrast to the *bsGainSetIndex* channel assignment for the “Duck other” effect.

User-supplied compression and boost factors shall be applied to all DRC sets except DRC sets with ducking, fading, or clipping effect.

In a drcCoefficientsUniDrcV1() payload, custom DRC characteristics can be defined to support more flexible gain modifications. These parametric characteristics can be used to describe the encoder-side DRC and the target characteristic. If a target characteristic is defined, it shall be applied after inverting the encoder-side characteristic. Hence, for the most general use, the encoder-side characteristic shall be invertible, i.e. have either a negative or positive slope for the entire gain range. Moreover, if the target characteristic has a section with constant gain, the encoder-side characteristic may also have constant gain in those sections. The parametric characteristics are computed as indicated in Table 19 and Table 20. Further details can be found in E.4.

If the only *drcSetEffect* is "clipping prevention", the clipping prevention gains are shifted depending on the target loudness to provide just enough signal attenuation to avoid clipping. The scaling factor is derived based on the *limiterPeakTarget*. See 6.10.2 for the calculation of *loudnessNormalizationGainDb*. The shifting mechanism will not be enabled if there are additionally defined effect bits apart from "clipping prevention".

Table 17 — Conversion of a DRC gain sample and associated slope from dB to linear domain (*slopeIsNegative*==1 if the source DRC characteristic has a negative slope)

```
toLinear (gainDb, slopeDb) {
    SLOPE_FACTOR_DB_TO_LINEAR = 0.1151f;        /* ln(10) / 20 */
    EFFECT_BIT_CLIPPING = 0x0100;               /* drcSetEffect 9 (Clip.Prev.) */
    EFFECT_BIT_FADE         = 0x0200;             /* drcSetEffect 10 (Fade) */
    EFFECT_BITS_DUCKING = 0x0400 | 0x0800;      /* drcSetEffect 11 or 12 (Ducking) */
    gainRatio = 1.0;
    if (((drcSetEffect & EFFECT_BITS_DUCKING) == 0) &&
                    (drcSetEffect != EFFECT_BIT_FADE) &&
                     (drcSetEffect != EFFECT_BIT_CLIPPING)) {
        if ((drcCharacteristicTarget > 0)   && (gainDb != 0.0)){
            gainRatio = mapGain(gainDb) / gainDb;     /* target characteristic from host */
        }
        else if (drcCoefficientsUniDrcV1Present == 1) {
            if (((gainDb > 0.0) && (slopeIsNegative == 1)) || ((gainDb < 0.0) && (slopeIsNegative == 0))) {
                if (targetCharacteristicLeftPresent == 1) {
                    gainRatio = mapGain(gainDb) / gainDb; /* target characteristic in payload */
                }
            }
            else if (((gainDb < 0.0) && (slopeIsNegative == 1)) || ((gainDb > 0.0) && (slopeIsNegative == 0))) {
                if (targetCharacteristicRightPresent == 1) {
                    gainRatio = mapGain(gainDb) / gainDb; /* target characteristic in payload */
                }
            }
        }
        if (gainDb < 0.0) {
        gainRatio *= compress;
        }
        else {
            gainRatio *= boost;
        }
    }
```

```
    if (gainScalingPresent) {
        if (gainDb < 0.0) {
            gainRatio *= attenuationScaling;
        }
        else {
            gainRatio *= amplificationScaling;
        }
    }
    if (duckingScalingPresent && (drcSetEffect & EFFECT_BITS_DUCKING)) {
        gainRatio *= duckingScaling;
    }
    gainLin = pow(2.0, gainRatio * gainDb / 6.0);
    slopeLin = SLOPE_FACTOR_DB_TO_LINEAR * gainRatio * gainLin * slopeDb;
    if (gainOffsetPresent) {
        gainLin *= pow(2, gainOffset/6.0);
    }
    /* The only drcSetEffect is "clipping prevention" */
    if (limiterPeakTargetPresent && (drcSetEffect == EFFECT_BIT_CLIPPING)) {
        gainLin *= pow(2, max(0.0, -limiterPeakTarget-loudnessNormalizationGainDb
                -loudnessNormalizationGainModificationDb)/6.0);
        if (gainLin >= 1.0) {
            gainLin = 1.0;
            slopeLin = 0.0;
        }
    }
    return (gainLin, slopeLin);
}
```

Table 18 — DRC gain mapping according to a target DRC characteristic

```
mapGain(gainQuant) {
    inLevel = inverseCompressorIO(gainQuant);
    outgain = targetCompressorIO(inLevel);
    return outgain;
}
```

Table 19 — Pseudo code to compute the DRC gain based on the input level in dB (*characteristicFormat*==0)

```
drcInputLoudnessTarget = -31;
compressorIO_sigmoidLeft(inLevelDb) {
    if (inLevelDb > drcInputLoudnessTarget) {
        printf("ERROR!");
        return (0.0);
    }
    tmp = (drcInputLoudnessTarget – inLevelDb) * ioRatioLeft;
    if (expLeft < INF) {                                    /* INF means infinity */
        outGainDb = tmp / pow(1 + pow(tmp/gainDbLeft, expLeft), 1/expLeft);
    } else {
        outGainDb = tmp;
```

```
    }
    if (flipSignLeft == 0) return outGainDb;
    else return -outGainDb;
}
compressorIO_sigmoidRight(inLevelDb) {
    if (inLevelDb < drcInputLoudnessTarget) {
        printf("ERROR!");
        return (0.0);
    }
    tmp = (drcInputLoudnessTarget – inLevelDb) * ioRatioRight;
    if (expRight < INF) {                                   /* INF means infinity */
        outGainDb = tmp / pow(1 + pow(tmp/gainDbRight, expRight), 1/expRight);
    } else {
        outGainDb = tmp;
    }
    if (flipSignRight == 0) return outGainDb;
    else return -outGainDb;
}
```

Table 20 — Pseudo code to compute the DRC gain based on the input level in dB (*characteristicFormat*==1)

```
drcInputLoudnessTarget = -31
compressorIO_nodesLeft(inLevelDb) {
    if (inLevelDb > drcInputLoudnessTarget) {
        printf("ERROR!");
        return (0.0);
    }
    nodeLevel[0] = drcInputLoudnessTarget;
    nodeGain[0] = 0.0;
    for (n=1; n<=characteristicNodeCount; n++) {
        nodeLevel[n] = nodeLevel[n-1] - nodeLevelDelta[n];
    }
    for (n=1; n<=characteristicNodeCount; n++) {
        if ((inLevelDb <= nodeLevel[n-1]) && (inLevelDb > nodeLevel[n])) {
            w = (nodeLevel[n]- inLevelDb)/(nodeLevel[n]-nodeLevel[n-1]);
            return (w * nodeGain[n-1] + (1.0-w) * nodeGain[n]);
        }
    }
    return (nodeGain[characteristicNodeCount]);
}
compressorIO_nodesRight(inLevelDb) {
    if (inLevelDb < drcInputLoudnessTarget) {
        printf("ERROR!");
        return (0.0);
    }
    nodeLevel[0] = drcInputLoudnessTarget;
    nodeGain[0] = 0.0;
    for (n=1; n<=characteristicNodeCount; n++) {
        nodeLevel[n] = nodeLevel[n-1] + nodeLevelDelta[n];
```

```
    }
    for (n=1; n<=characteristicNodeCount; n++) {
        if ((inLevelDb >= nodeLevel[n-1]) && (inLevelDb < nodeLevel[n])) {
            w = (nodeLevel[n]- inLevelDb)/(nodeLevel[n]-nodeLevel[n-1]);
            return (w * nodeGain[n-1] + (1.0-w) * nodeGain[n]);
        }
    }
    return (nodeGain[characteristicNodeCount]);
}
```

The gain interpolation is implemented by the pseudo code in Table 21. Dependent on the configuration variable *gainInterpolationType*, either spline interpolation or linear interpolation is performed. The input variables are:

— the time difference between the two gain samples in units of the target sample rate interval *tGainStep*

— a pair of subsequent gain samples *gain0* and *gain1* in dB

— a pair of corresponding slope steepness values *slope0* and *slope1* in the dB domain. These are neglected for linear interpolation.

This function uses toLinear() to convert the variables to the linear domain. The result is a smooth sequence of gain values at the target sample rate located between the pair of gain samples. The target sample rate is the sample rate of the compressed audio signal.

Table 21 — Interpolation of the DRC gain for one spline or linear segment

```
interpolateDrcGain(tGainStep, gain0, gain1, slope0, slope1)
{
    int n;
    float k1, k2, a, b, c, d;

    float slopeLeft;
    float slopeRight;
    float gainLeft;
    float gainRight;

    (gainLeft, slopeLeft) = toLinear (gain0, slope0);
    (gainRight, slopeRight) = toLinear (gain1, slope1);

    if (gainInterpolationType == 0) {
        slopeLeft = slopeLeft / deltaTmin;
        slopeRight = slopeRight / deltaTmin;

        bool useCubicInterpolation = TRUE;
        int tConnect;
        float tConnectFloat;
        if (abs(slopeLeft) > abs(slopeRight)) {
            tConnectFloat = 2.0 * (gainRight - gainLeft - slopeRight * tGainStep) / (slopeLeft - slopeRight);
            tConnect = (floor) (0.5 + tConnectFloat);
            if ((tConnect >= 0) && (tConnect < tGainStep)) {
```

```
                useCubicInterpolation = FALSE;
                result[0] = gainLeft;
                result[tGainStep] = gainRight;
                a = (slopeRight - slopeLeft) / (tConnectFloat + tConnectFloat);
                b = slopeLeft;
                c = gainLeft;
                for (n=1; n<tConnect; n++) {
                    float t = (float) n;
                    result[n] = (a * t + b ) * t + c;
                    result[n] = max (0.0, result[n]);
                }
                a = slopeRight;
                b = gainRight;
                for (     ; n<tGainStep; n++) {
                    float t = (float) (n - tGainStep);
                    result[n] = a * t + b;
                }
            }
        }
        else if (abs(slopeLeft) < abs(slopeRight)) {
            tConnectFloat = 2.0 * (gainLeft - gainRight + slopeLeft * tGainStep) / (slopeLeft - slopeRight);
            tConnectFloat = tGainStep - tConnectFloat;
            tConnect = (floor) (0.5 + tConnectFloat);
            if ((tConnect >= 0) && (tConnect < tGainStep)) {
                useCubicInterpolation = FALSE;
                result[0] = gainLeft;
                result[tGainStep] = gainRight;
                a = slopeLeft;
                b = gainLeft;
                for (n=1; n<tConnect; n++) {
                    float t = (float) n;
                    result[n] = a * t + b;
                }
                a = (slopeRight - slopeLeft) / (2.0 * (tGainStep - tConnectFloat));
                b = - slopeRight;
                c = gainRight;
                for (     ; n<tGainStep; n++) {
                    float t = (float) (tGainStep-n);
                    result[n] = (a * t + b ) * t + c;
                    result[n] = max (0.0, result[n]);
                }
            }
        }
        if (useCubicInterpolation == TRUE) {
            float tGainStepInv = 1.0 / (float)tGainStep;
            float tGainStepInv2 = tGainStepInv * tGainStepInv;

            k1 = (gainRight - gainLeft) * tGainStepInv2;
            k2 = slopeRight + slopeLeft;
            a = tGainStepInv * (tGainStepInv * k2 - 2.0 * k1);
```

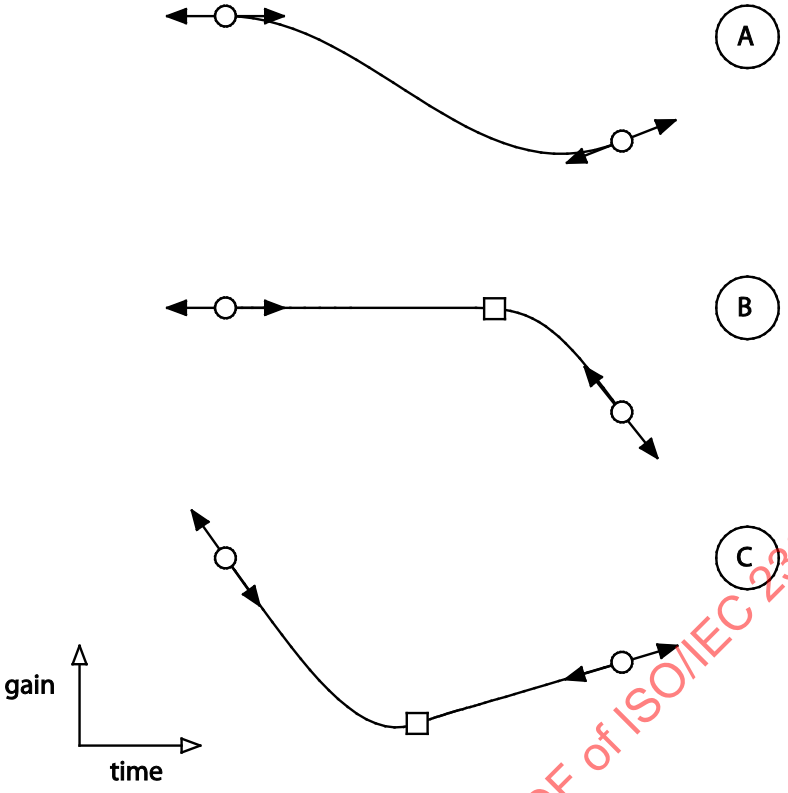
```
                b = 3.0 * k1 - tGainStepInv * (k2 + slopeLeft);
                c = slopeLeft;
                d = gainLeft;
                result[0] = gainLeft;
                result[tGainStep] = gainRight;
                for (n=1; n<tGainStep; n++) {
                    float t = (float) n;
                    result[n] = (((a * t + b ) * t + c ) * t ) + d;
                    result[n] = max (0.0, result[n]);
                }
            }
        }
        else {
            a = (gainRight – gainLeft) / tGainStep;
            b = gainLeft;

            result[0] = gainLeft;
            result[tGainStep] = gainRight;

            for (n=1; n<tGainStep; n++) {
                float t = (float)n;
                result[n] = a * t + b;
            }
        }
        return result;
}
```

6.4.7 Spline interpolation

Interpolation of the DRC gain in the decoder is based on pairs of gain samples. For spline interpolation, in addition to the node coordinates (time and value in dB), also the slope information is given for each pair. The decoder will choose one of three available types of interpolation as illustrated in Figure 3. In most cases, cubic interpolation is chosen. However, under certain conditions, a hybrid interpolation combining linear and quadratic interpolation is applied instead. For the hybrid interpolation, a node is inserted between the two given nodes (shown as a square). At one side of that node, linear interpolation is applied and quadratic interpolation is applied at the other. The method is fully specified in 6.4.6.



NOTE A square indicates the connection point of the linear and quadratic sub-segment.

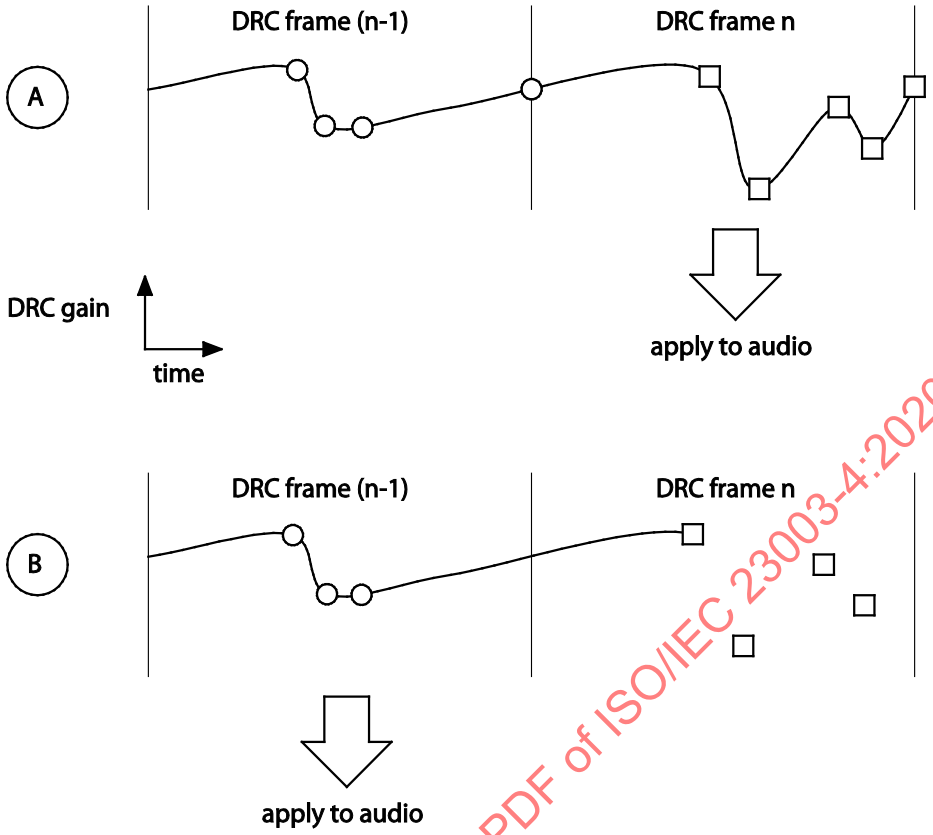
Figure 3 — Spline segment types: cubic interpolation (A), hybrid using linear and quadratic interpolation (B, C)

6.4.8 Look-ahead in decoder

The DRC tool decoder can be operated in one of two delay modes. The low-delay mode immediately applies the decoded DRC gain while the default mode applies the DRC gain with a delay of one DRC frame. The low-delay mode requires that the flag *fullFrame* is 1, which signals that a gain value sample is located at the end of each DRC frame. For the default mode, the flag *fullFrame* can also be 0. Then it supports gain sample interpolation from any position of the current DRC frame to any position of the next DRC frame.

Figure 4 illustrates the two delay modes. The upper diagram shows that each DRC frame has a spline node at the end of the frame, so that the entire DRC gain curve for that frame can immediately be generated by interpolation. The lower diagram shows that the interpolated gain curve is applied with a delay of one DRC frame, since the interpolation for frame *n*-1 can only be completed after the first node of frame *n* is received.

For common perceptual codecs, the default delay mode does not require additional decoder delay. The delay is already required due to the overlap-add operation. The decoder appends audio samples with zeros at the end to reconstruct the last frame. If the DRC gains for the last frame do not cover the entire duration, the last gain shall be repeated until the end of the frame. If DRC gains are required for the first frame in default delay mode, they should be generated based on a node at the beginning of the first frame with 0 dB gain and, when in spline interpolation mode, with a slope of 0.



Key

circles spline nodes received in frame *n*-1

squares spline nodes received in frame *n*

NOTE The solid line illustrates the interpolated DRC gain up to DRC frame *n*.

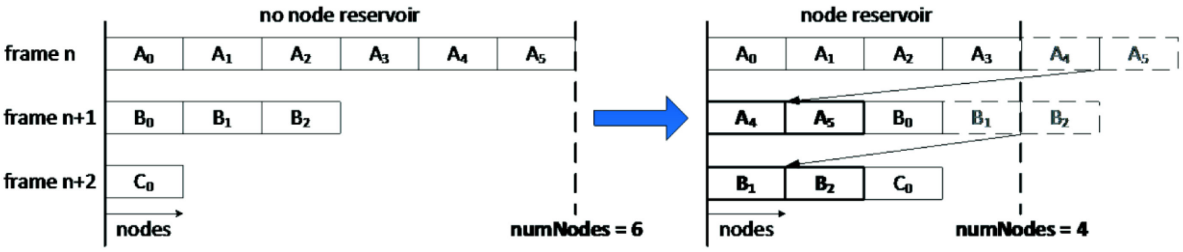
Figure 4 — Delay modes: low-delay and *fullFrame* = 1 (A), default delay and *fullFrame* = 0 (B).

The low-delay mode is suitable for decoders that do not have inherent delay such as a delay due to overlap-add. For instance, this is the case for some lossless codecs. For any PCM audio format, low-delay mode shall be used.

For ISO/IEC 14496 (ISO base media file format), the delay mode is explicitly conveyed in the drcCoefficientsUniDrc() payload in the *delayMode* field (see Table 69). The encoding of *delayMode* is given in Table A.20.

6.4.9 Node reservoir

To reduce bitrate peaks, a so-called “node reservoir” can be employed. The node reservoir can be only applied in the default delay mode, where the DRC gain is applied with a delay of one DRC frame (see 6.4.8). In addition, the *fullFrame* flag shall be set to 0. The node reservoir mechanism shifts nodes from the current frame into the subsequent frame (see Figure 5). In order to keep the decoding delay constant, it is not allowed to shift the first node of a frame since in default delay mode, the first node is needed to fully decode the previous frame. Thus, the maximum number of nodes that can be shifted to the subsequent frame is limited to *nNodes-1* nodes.



NOTE A_x, B_x, and C_x are nodes regularly transmitted in DRC frame n, n+1, and n+2, respectively (left). When the node reservoir is used, some nodes (bold) are transmitted with a delay of one frame (right).

Figure 5 — Illustration of node reservoir mechanism

Each node is defined by time, gain, and slope data (*gainInterpolationType* = 0). The gain and slope data have chronological order in the bitstream, therefore the data from the node reservoir precedes the regular nodes of a particular frame in the bitstream. During decoding, the nodes of the node reservoir can be identified because their time values are encoded with an offset of *drcFrameSize*. Therefore, the use of the node reservoir is not explicitly signalled. The offset is subtracted in the decoding process to obtain the correct time value.

The number of nodes shifted to the node reservoir in each frame is not transmitted explicitly, but results implicitly from the number of time values larger than *drcFrameSize*. The decoding of time values precedes the decoding of gain and slope values. Therefore, the assignment of gain and slope values to time values is known after decoding the time values. The decoding process is fully specified in 6.4.5 and 6.4.10.

6.4.10 Applying the compression

The interpolated gain values of each interpolation segment are concatenated to generate a complete gain vector *gain*[*g*][*b*][*t*] for the entire DRC frame. Finally, the gain vector is applied as shown in Table 22. The function *channelInDrcGroup*() returns TRUE if the current channel *c* belongs to the current DRC channel group. The scheduling of the interpolation segments depends on the delay mode as indicated in Table 22.

Table 22 — Concatenation of interpolation segments to a gain vector and application of the DRC gain vector to the audio channels

```
for(g=0; g<nDrcChannelGroups; g++) {
  for(b=0; b<nDrcBands[g]; b++) {
    if (delayMode == DELAY_MODE_DEFAULT) {
      nNodesNodeReservoir = 0;
      for (k=0; k<nNodes[g][b]; k++) {
        if (tDRC[g][b][k] >= drcFrameSize) {
          nNodesNodeReservoir++;
        }
      }
      for (k=0; k<nNodesNodeReservoir; k++) {
        tDRCprev[g][b][nNodesPrev[g][b]+k]=tDRC[g][b][nNodes[g][b]-nNodesNodeReservoir+k]-drcFrameSize;
        gDRCprev[g][b][nNodesPrev[g][b]+k]=gDRC[g][b][k];
        sDRCprev[g][b][nNodesPrev[g][b]+k]=sDRC[g][b][k];
      }
      memmove(&gDRC[g][b][0],&gDRC[g][b][nNodesNodeReservoir], nNodes[g][b]-nNodesNodeReservoir);
```

```
    memmove(&sDRC[g][b][0],&sDRC[g][b][nNodesNodeReservoir], nNodes[g][b]-nNodesNodeReservoir);
    nNodesPrev[g][b] = nNodesPrev[g][b] + nNodesNodeReservoir;
    nNodes[g][b] = nNodes[g][b] – nNodesNodeReservoir;
    for (k=0; k<nNodesPrev[g][b]-1; k++) {
      duration = tDRCprev[g][b][k+1] – tDRCprev[g][b][k];
      interpolationSegment= interpolateDrcGain(duration, gDRCprev[g][b][k],
                  gDRCprev[g][b][k+1], sDRCprev[g][b][k], sDRCprev[g][b][k+1]);
      for (t=0; t<duration; t++) {
        gain[g][b][t+tDRCprev[g][b][k]] = interpolationSegment[t];
      }
    }
    k = nNodesPrev[g][b]-1;
    duration = drcFrameSize + tDRC[g][b][0] – tDRCprev[g][b][k];
    interpolationSegment = interpolateDrcGain(duration, gDRCprev[g][b][k],
        gDRC[g][b][0], sDRCprev[g][b][k], sDRC[g][b][0]]);
    for (t=0; t<duration; t++) {
      gain[g][b][t+tDRCprev[g][b][k]] = interpolationSegment[t];
    }
  }
  else {
    k = nNodesPrev[g][b]-1;
    duration = tDRC[g][b][0]+1;
    interpolationSegment = interpolateDrcGain(duration, gDRCprev[g][b][k],
        gDRC[g][b][0], sDRCprev[g][b][k], sDRC[g][b][0]]);
    for (t=1; t<=duration; t++) {
      gain[g][b][t-1] = interpolationSegment [t];
    }
    for (k=0; k<nNodes[g][b]-1; k++) {
      duration = tDRC[g][b][k+1] – tDRC[g][b][k];
      interpolationSegment = interpolateDrcGain(duration, gDRC[g][b][k],
        gDRC[g][b][k+1], sDRC[g][b][k], sDRC[g][b][k+1]);
      for (t=1; t<=duration; t++) {
        gain[g][b][t+tDRC[g][b][k]] = interpolationSegment [t];
      }
    }
  }

  /* Apply gain to DRC bands of audio in each channel */

  for (c=0; c<nChannels; c++) {
    if (channelInDrcGroup(c, g)) {
      for (t=0; t<drcFrameSize; t++) {
        audioBandOut[c][b][t] = audioBandIn[c][b][t] * gain[g][b][t];
      }
    }
  }
  if (delayMode == DELAY_MODE_DEFAULT) {
    for (k=0; k<nNodes; k++) {
      gDRCprev[g][b][k] = gDRC[g][b][k];
      sDRCprev[g][b][k] = sDRC[g][b][k];
```

```
            tDRCprev[g][b][k] = tDRC[g][b][k];
          }
          nNodesPrev[g][b] = nNodes[g][b];
          for (t=0; t<drcFrameSize; t++) {
            gain[g][b][t] = gain[g][b][t + drcFrameSize];
          }
        }
        else {
          k=nNodes[g][b]-1;
          gDRCprev[g][b][k] = gDRC[g][b][k];
          sDRCprev[g][b][k] = sDRC[g][b][k];
          nNodesPrev[g][b] = nNodes[g][b];
        }
      }
    }
    for(g=0; g<nDrcChannelGroups; g++) {
      for(c=0; c<nChannels; c++) {
        if (channelInDrcGroup(c, g)) {
          for (t=0; t<drcFrameSize; t++) {
            sum = 0.0;
            for(b=0; b<nDrcBands[g]; b++) {
              sum = sum + audioBandOut[c][b][t];
            }
            audioSampleOut[c][t] = sum;
          }
        }
      }
    }
```

Table 22 is based on the following assumption:

— *interpolationSegment* is a vector that contains the gain values of one interpolation segment.

— *duration* is an integer number describing the duration of the interpolation segment in units of audio sample intervals.

— *nNodes* is the number of gain values in the current DRC frame.

— *drcFrameSize* is the number of audio sample intervals in a DRC frame.

— Initialization of the following variables:

```
gDRCprev[g][b][0]=0.0,
sDRCprev[g][b][0]=0.0,
nNodesPrev[g][b]=1,
if (timeAlignment==0)
      tDRCprev[g][b][0]=drcFrameSize-1
else
      tDRCprev[g][b][0]=drcFrameSize-timeDeltaMin+floor((timeDeltaMin-1)/2).
```

6.4.11 Dynamic equalization

6.4.11.1 Overview

Shaping filters are used to dynamically modify the audio signal spectrum as the DRC gain changes, i.e. the filters provide a kind of dynamic equalization. Figure 6 shows the shaping filters for time-domain processing that includes filters to cut and boost the low and high frequency range. The filter coefficients are dynamically adapted depending on the final DRC gain value after interpolation. If the gain value is below 0 dB, the filters for cutting have a flat frequency response. In contrast, when the gain value is above 0 dB, the filters for boosting have a flat frequency response. The filter coefficient adaptation is controlled by the parameters for corner frequency and filter strength.

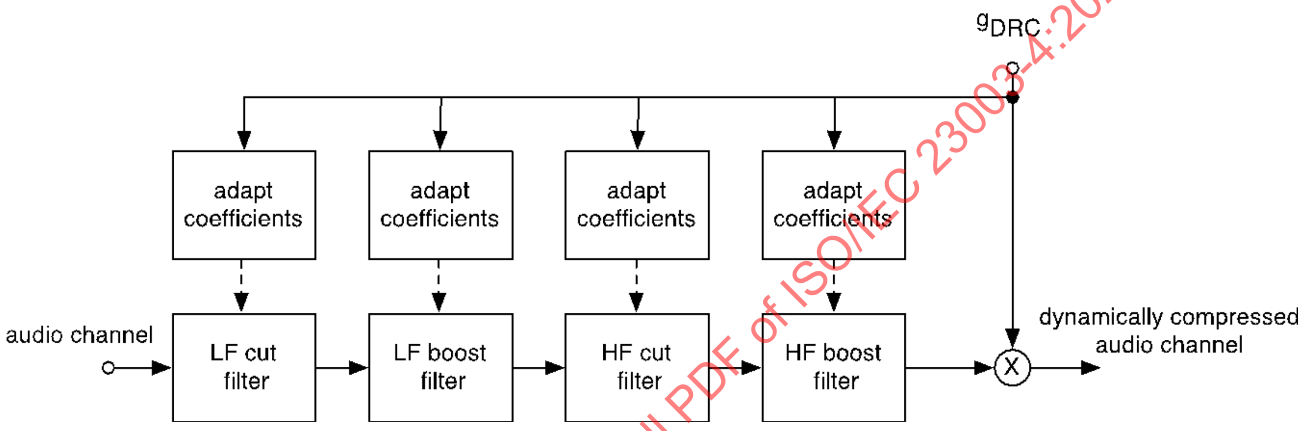


Figure 6 — Bank of spectral shaping filters

Shaping filters are only applied for single-band DRCs and if the DRC operates in the time domain. They are ignored in the QMF domain. If a similar effect is desired for operation in the QMF domain, it is recommended to use a multi-band DRC instead. The application of shaping filters introduces a phase shift of the audio signal. Hence, if the shaping filters of the channel groups differ, the output channel groups may not be phase aligned anymore. Therefore, it is recommended to apply the same shaping filter bank to all channel groups, unless the audio signal groups are uncorrelated.

Each filter has one adaptive coefficient, which is dynamically computed based on the linear DRC gain value g_{DRC} (after interpolation) as indicated by the formula for y_1. The linear gain value is first warped according to the pseudo code for warpGain() in Table 23.

$$g_{\mathrm{warped}} = \mathrm{warpGain}\left(g_{\mathrm{DRC}}, \mathrm{mode}\right)$$

$$y_1 = \begin{cases} x_1 & ; \quad g_{\mathrm{warped}} \le 0 \\ x_1 + \left(y_{1,\mathrm{bound}} - x_1\right)\dfrac{g_{\mathrm{warped}}}{g_{\mathrm{warped,max}}} & ; \quad 0 < g_{\mathrm{warped}} < g_{\mathrm{warped,max}} \\ y_{1,\mathrm{bound}} & ; \quad \mathrm{else} \end{cases}$$

with $y_{1,\mathrm{bound}}$ as defined in Table A.31 and Table A.32, and

$$g_{\mathrm{warped,max}} = \mathrm{warpGain}\left(g_{\mathrm{DRC,max}}, \mathrm{'cut'}\right) \quad \text{with} \quad g_{\mathrm{DRC,max}} = 10^{32/20}$$

The warping function is defined in Table 23 using the gain offset parameter g_{offset} as defined in Table A.33 and Table A.34.

Table 23 — Pseudo code of warpGain() function

```
warpGain(gDrc, mode) {
    switch(mode) {
        case 'cut':     /* LF or HF cut filter */
            if (gDrc <= 1.0) return (-1.0);
            else return (gDrc-1)/(gDrc-1+gOffset);
        case 'boost':     /* LF or HF boost filter */
            if (gDrc >= 1.0) return (-1.0);
            else return (1-gDrc)/(1+gDrc*(gOffset-1));
    }
}
```

6.4.11.2 Adaptation of shaping filters

All shaping filters are adapted at the first audio sample of each output audio frame and whenever the DRC gain change compared to the previous adaptation exceeds the threshold according to the following formula:

$$\mathrm{abs}\left(g_{\mathrm{DRC}} - g_{\mathrm{DRC,last}}\right) > 0.0001\ g_{\mathrm{DRC,last}}$$

where $g_{\mathrm{DRC,last}}$ is the DRC gain value when the shaping filters were last updated. On initialization, the value is set to $g_{\mathrm{DRC,last}} = 1.0$.

6.4.11.3 Low-frequency shaping filters

Each low-frequency filter is a first order IIR filter with real coefficients of the form:

$$H_{\mathrm{LF}}(z) = \frac{1 + b_1 z^{-1}}{1 + a_1 z^{-1}}$$

The coefficient x_1 is defined depending on the radius, r, in Table A.35 as

$$x_1 = -r$$

The filter coefficients for the LF filters are assigned as shown in Table 24.

Table 24 — Assignment of filter coefficients for LF shaping filters

Filter type	Filter coefficients	
LF cut filter	$a_1 = x_1$	$b_1 = y_1$
LF boost filter	$a_1 = y_1$	$b_1 = x_1$

6.4.11.4 High-frequency shaping filters

Each high-frequency filter is a second order IIR filter with real coefficients of the form:

$$H_{\mathrm{HF}}(z) = g_{\mathrm{norm}} \frac{1 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \qquad g_{\mathrm{norm}} = \frac{1 + a_1 + a_2}{1 + b_1 + b_2}$$

The corner frequency of the filter depends on the audio sample rate f_s and the normalized corner frequency $f_{c,\mathrm{norm}}$:

$$f_c = f_{c,\mathrm{norm}} f_s$$

The fixed coefficients for both the HF cut and boost filter depend on the corner frequency $f_{c,\mathrm{norm}}$ and the pole/zero radius parameter r (see Table A.36).

$$x_1 = -2r\cos\left(2\pi f_{c,\mathrm{norm}}\right)$$

$$x_2 = r^2$$

$$y_2 = x_2$$

The filter coefficients for the HF filters are assigned as shown in Table 25.

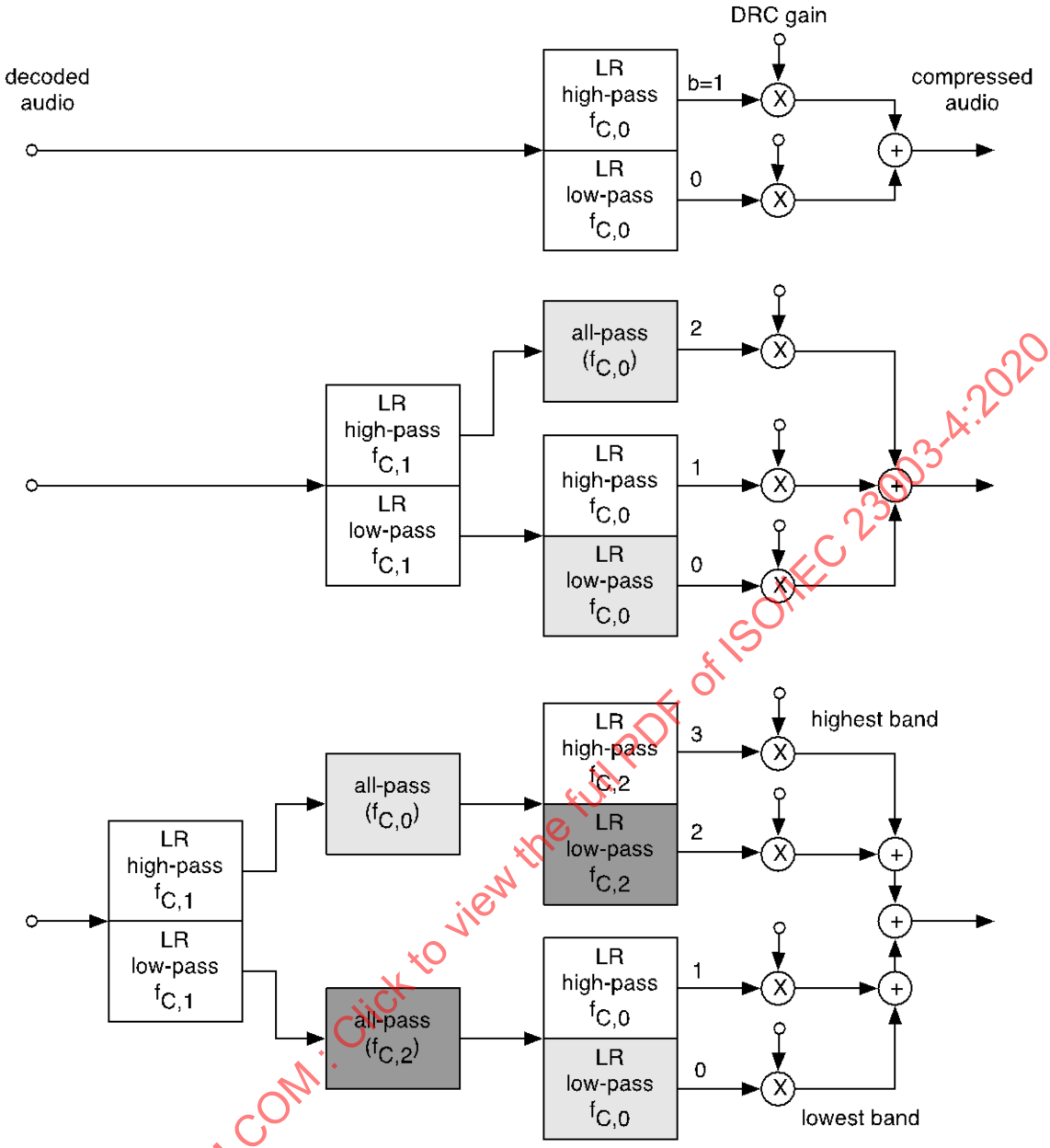
Table 25 — Assignment of filter coefficients for HF shape filters

Filter type	Filter coefficients			
HF cut filter	$a_1 = x_1$	$a_2 = x_2$	$b_1 = y_1$	$b_2 = y_2$
HF boost filter	$a_1 = y_1$	$a_2 = y_2$	$b_1 = x_1$	$b_2 = x_2$

6.4.12 Multi-band DRC filter bank

When the DRC gains are applied in the time domain and a multi-band DRC is used, the time-domain audio signal shall be split into sub-bands before the DRC gains are applied to the bands. The filter configuration parameters are conveyed by the drcCoefficientsUniDrc() defined in Table 67, Table 68 or Table 69. The tables provide the bitstream syntax for the number of bands and the crossover frequency indices between bands. The number of time-domain DRC bands cannot be larger than four.

The time-domain audio signal is split into the specified number of bands by Linkwitz-Riley (LR) filters with a topology shown in Figure 7. If there are more than two bands, all-pass filters are added to compensate for the delay of the different outputs, so that they are all in phase. The low-pass and high-pass filters are implemented as second order sections (biquads).



NOTE The band index b increases with the frequency of the band. The crossover frequencies $f_{C,b}$ increase with index b, i.e. $f_{C,b+1} > f_{C,b}$ Crossover frequencies in brackets of an all-pass filter specify the corresponding LR low-pass filter with the matching phase response.

Figure 7 — Topology of Linkwitz-Riley crossover filters for 2, 3, and 4 bands

Each Linkwitz-Riley crossover filter is composed of a pair of a complementing low-pass and high-pass filter that results in a flat frequency response overall. Each LR low-pass filter is created by a cascade of two identical second-order Butterworth (BW) low-pass filters. Similarly, each LR high-pass filter is a cascade of two identical BW high-pass filters with the same order and cutoff frequency as the BW low-pass filters. Each all-pass filter is a second order IIR filter.

Each BW filter and each all-pass filter is implemented as second order section with the following transfer function.

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{3}$$

Based on the crossover frequency indices in Table A.8, the decoder can look up the normalized crossover frequencies $f_{c,Norm}$ or the filter coefficient parameters γ and δ. The filter coefficients are then computed using Table 26 for the BW filters and Table 27 for the all-pass filters. The crossover frequencies f_c in Hz are computed by:

$$f_c = f_s \cdot f_{c,Norm} \tag{4}$$

Table 26 — Butterworth filter coefficient formulas

	BW low-pass	BW high-pass
Normalized cutoff frequency	$\omega_0 = \tan(\pi f_{c,Norm})$	
Intermediate parameters	$\delta = \frac{1}{1 + \sqrt{2\omega_0 + \omega_0^2}}$ $\gamma = \omega_0^2 \delta$	
Final BW filter coefficients	$a_{LP,0} = 1$ $a_{LP,1} = 2(\gamma - \delta)$ $a_{LP,2} = 2(\gamma + \delta) - 1$ $b_{LP,0} = \gamma$ $b_{LP,1} = 2\gamma$ $b_{LP,2} = \gamma$	$a_{HP,0} = 1$ $a_{HP,1} = 2(\gamma - \delta)$ $a_{HP,2} = 2(\gamma + \delta) - 1$ $b_{HP,0} = \delta$ $b_{HP,1} = -2\delta$ $b_{HP,2} = \delta$

The all-pass filters in Figure 7 are used to generate the same phase response as one of the LR low-pass filters (with matching gray level and matching f_c in Figure 7) so that the signals of all bands are in phase at the output of the filter bank. The all-pass filter coefficients are derived from the coefficients of the corresponding BW low-pass filter as shown in Table 27.

Table 27 — All-pass filter coefficient formulas

$a_{AP,0} = a_{LP,0}$
$a_{AP,1} = a_{LP,1}$
$a_{AP,2} = a_{LP,2}$
$b_{AP,0} = a_{LP,2}$
$b_{AP,1} = a_{LP,1}$
$b_{AP,2} = a_{LP,0}$

After the DRC gains are applied to the individual bands, the final audio signal is computed by adding all bands.

In configurations that apply different multiband filters to different channel groups, additional all-pass filters shall be incorporated to achieve an in-phase output of all channel groups. The example in Figure 8 shows three channel groups that are processed with a single band (0), two DRC bands (1), and four DRC bands (2). The overall phase response of the system is determined by the following steps:

— Construct a filter structure for each channel group according to the crossover frequencies and number of bands.

— Concatenate all filter structures to determine the target phase response.

— Add all-pass filters to each channel group filter structure to match the target phase response.

— Eliminate identical all-pass filters that appear in all channel groups to minimize the delay if possible while maintaining in-phase output of all channel groups.

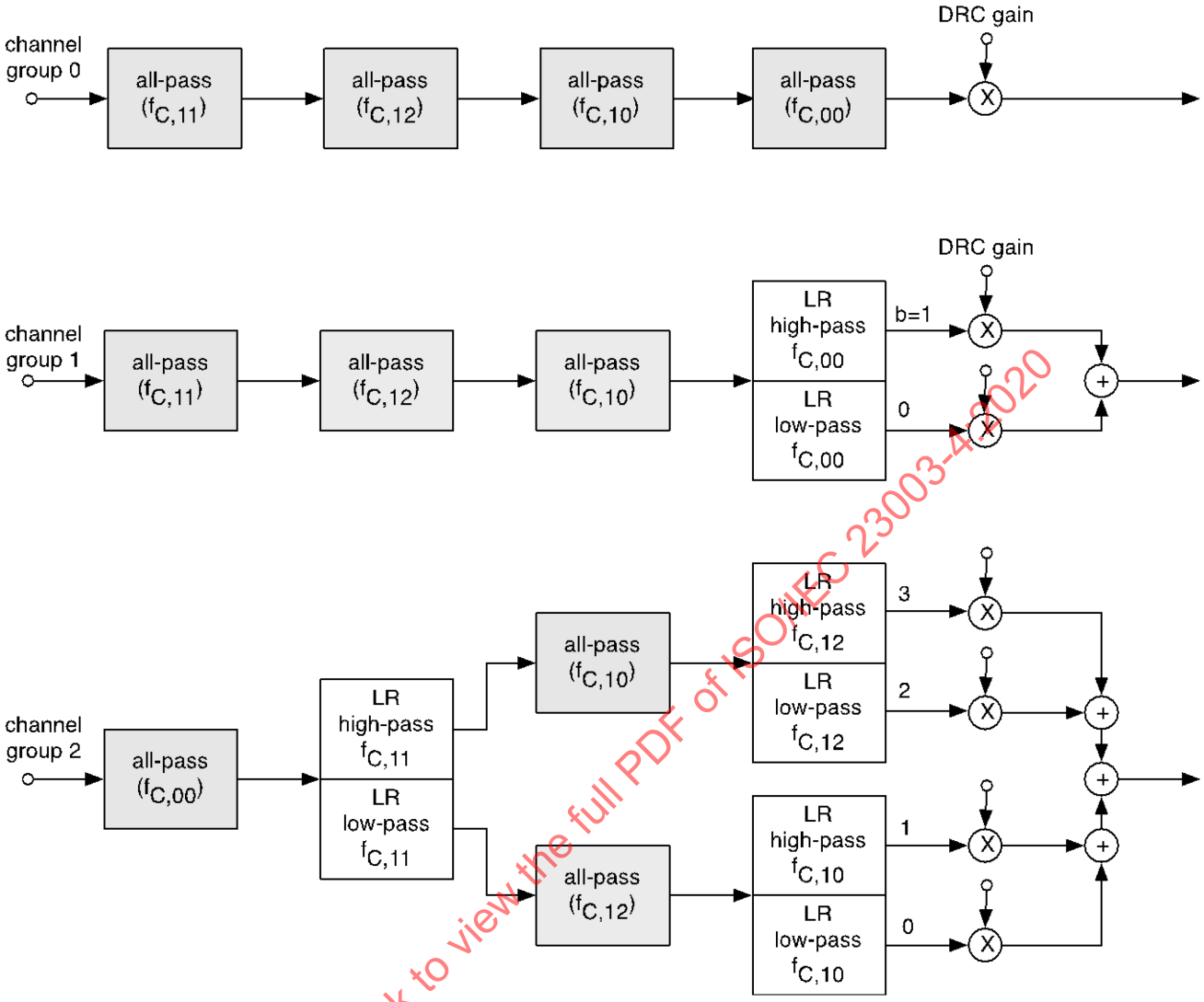


Figure 8 — Example of phase adjustments for time-domain multiband DRC with different DRC bands for each channel group

Support for up to three different time-domain multiband DRC filter banks with phase alignment of the corresponding channel groups is mandatory. This number includes full-band DRCs and counts each multiband filter bank with a different crossover frequency. If two DRC sets for time-domain application are combined using the *dependsOnDrcSet* field, not more than one of them can be a multiband DRC. In this case, the exact output waveform depends on the processing order of the DRC sets, which is defined in 6.3.5. However, if the multiband DRC set has a *downmixId* of 0x7F, the information whether it is processed before or after the downmix needs to be provided to fully define the output waveform, e.g. for conformance testing.

Systems that process audio objects shall not apply time-domain multi-band DRC to only a sub-set of correlated objects since that will result in phase misalignment of the objects in the final mix.

6.5 Sub-band domain DRC

While the application of DRC gains in the time-domain is mandatory for MPEG AAC up to four DRC bands, other MPEG codecs might use sub-band domain DRC. The concept of sub-band domain DRC means that the existing sub-band signals of the audio decoder are subject to the DRC gain application. Therefore, it is not necessary to add time-domain band splitting for a multi-band DRC and it is possible

to apply DRC gains before rendering and/or downmixing in the frequency domain. Table 28 contains a non-exhaustive list of codecs and the domain where the DRC gain is applied. The domain depends on the decoder configuration and not on the bitstream. For instance, if MPEG-Surround is decoded with a plain AAC decoder, the DRC gains are applied in the time domain. Furthermore, the sub-band domain is not the MDCT domain of a core codec – it is usually the QMF domain. The applicable audio codec standard may specify the DRC application in more detail (see also Annex C).

Table 28 — Domain of DRC gain applications for various MPEG decoders

Decoder	Time-domain DRC	Sub-band DRC
MPEG-4 AAC	✓	-
MPEG-4 HE-AAC	-	✓
MPEG-D Surround	-	✓
MPEG-D SAOC	-	✓
MPEG-D USAC	✓	✓
MPEG-H	✓	✓

In the most general case, the DRC gains are decoded as described in 6.4.5. They are interpolated using the same technique as described in Table 21 and Table 22, however, the sampling rate of the interpolation result is lowered to match the sample rate of the sub-band signals. This can be achieved by sub-sampling the interpolated time-domain DRC gains by a factor of L or by directly interpolating using the sub-band sample rate as target.

In many cases of sub-band DRC, it is more efficient to override the default *deltaTmin* in the encoder by setting *timeDeltaMin* to a value that matches the sub-band sample interval. With that, unnecessary interpolation and downsampling can be avoided in the decoder and the DRC time resolution is not higher than necessary (see also D.1.2).

To reproduce the frequency response of the Linkwitz-Riley filters, the filter slopes are generated in the sub-band domain as illustrated by the pseudo code in Table 30 for *bandType*==1. The preliminary weighting coefficients reflect the filter responses including the slopes. The final weighting coefficients are obtained by normalizing the preliminary weighting coefficents, such that the sum of the weights of all DRC band gains at each decoder sub-band centre frequency is 1.0.

The down-sampling and delay operations can be expressed by the first part of pseudo code in Table 29. The remaining part shows how the weighting coefficients are applied to the DRC gains to generate the gains for the audio decoder sub-bands. The meaning of variables and functions of the pseudo code is explained in Table 31. The description assumes that the sample rates in all sub-bands are equal. If this is not the case, the down-sampling factor L needs to be adjusted for the different sub-band sample rates.

Table 29 — DRC gain down-sampling, overlap, and application in decoder sub-bands

```
/* resample DRC gain */
for (g=0; g<nDrcChannelGroups; g++) {
    for (b=0; b<nDrcBands[g]; b++) {
        for (m=0; m<drcFrameSizeSb; m++) {
            gainLr[g][b][m]=gain[g][b][(m*L+floor((L-1)/2)];
        }
    }
}
for (g=0; g<nDrcChannelGroups; g++) {
    if (nDrcBands[g] == 1) {
        for (s=0; s<nDecoderSubbands; s++) {
            overlapWeight[g][0][s] = 1.0;
        }
    } else {
        overlapWeight[g] = generateOverlapWeights(g)
    }
    for (b=0; b<nDrcBands[g]; b++) {
        for (s=0; s<nDecoderSubbands; s++) {
            for (m=0; m<drcFrameSizeSb; m++) {
                gainSb[g][s][m] += overlapWeight[g][b][s] * gainLr[g][b][m];
            }
        }
    }
}
/* apply DRC gain in sub-bands */
for (g=0; g<nDrcChannelGroups; g++) {
    for (c=0; c<nChannels; c++) {
        if (channelInDrcGroup(c, g)) {
            for (s=0; s<nDecoderSubbands; s++) {
                for (m=0; m<drcFrameSizeSb; m++) {
                    audioSampleSbOut[c][s][m]=gainSb[g][s][m]*audioSampleSbIn[c][s][m];
                }
            }
        }
    }
}
```


Table 30 — Computation of overlap weigths

```
generateSlope(fCrossNormLo, fCrossNormHi) {
    float filterSlope = -24.0;              /* filter slope in dB per octave /
    float log10_2_inv = 3.32192809;    /* 1.0 / log10(2) */
    float norm = 0.05 * filterSlope * log10_2_inv;
    for (s=0; s<nDecoderSubbands; s++) {
        if (fCenterNormSb[s]<fCrossNormLo) {
            response[s] = pow (10.0, norm * log10(fCrossNormLo / fCenterNormSb[s]));
        }
        else if (fCenterNormSb[s]<fCrossNormHi) {
```

```
                response[s] = 1.0;
            }
            else {
                response[s] = pow (10.0, norm * log10(fCenterNormSb[s] / fCrossNormHi));
            }
        }
        return (response)
}

generateOverlapWeights(g) {
        if (drcBandType[g] == 1) {
            fCrossNormLo = 0.0f;
            for (b=0; b<nDrcBands[g]; b++) {
                if (b<nDrcBands - 1) {
                    fCrossNormHi = fCross[g][b+1];
                }
                else {
                    fCrossNormHi = 0.5;
                }
                overlapWeight[b] = generateSlope (fCrossNormLo, fCrossNormHi);
                fCrossNormLo = fCrossNormHi;
            }
            for (s=0; s<nDecoderSubbands; s++) {
                wNorm[s] = overlapWeight[0][s];
                for (b=1; b<nDrcBands[g]; b++) {
                    wNorm[s] += overlapWeight[b][s];
                }
            }
            for (s=0; s< nDecoderSubbands; s++) {
                for (b=0; b<nDrcBands[g]; b++) {
                    overlapWeight[b][s] /= wNorm[s];
                }
            }
        } else {
            startSubBandIndex = 0;
            for (b=0; b<nDrcBands[g]; b++) {
                if (b < nDrcBands[g]-1) {
                    stopSubBandIndex = startSubBandIndex[g][b+1]-1;
                } else {
                    stopSubBandIndex = nDecoderSubbands-1;
                }
                for (s=0; s< nDecoderSubbands; s++) {
                    if (s >= startSubBandIndex && s <= stopSubBandIndex) {
                        overlapWeight[b][s] = 1.0;
                    } else {
                        overlapWeight[b][s] = 0.0;
                    }
                }
                startSubBandIndex = stopSubBandIndex+1;
            }
```

```
    }
    return (overlapWeight);
}
```

Table 31 — Explanation of pseudo code items

Code item	Meaning
gainSb	DRC gain to be applied to decoder sub-bands
gainLr	Low-rate (resampled) DRC gain
fCross	Normalized crossover frequency
startSubBandIndex	Sub-band start index of multiband DRC band
drcFrameSizeSb	Number of sub-band samples per sub-band in one audio frame
nDecoderSubbands	Number of audio decoder sub-bands
fCenterNormSb	Center frequency of audio decoder sub-band
audioSampleSbIn	Decoded sub-band audio sample before dynamic compression
audioSampleSbOut	Decoded sub-band audio sample after dynamic compression

6.6 Generation of DRC gain values at the decoder

In addition to dynamic DRC gains included in the uniDrcGain() payload, the DRC tool alternatively allows to generate time-varying DRC gain values at the decoder based on transmitted parametric DRC configurations. The parametric DRC configurations are conveyed as an extension to the static uniDrcConfig() payload (see Table 75 and Table A.12).

6.6.1 Overview

Parametric DRC payloads are transmitted in the uniDrcConfigExtension() payload. The following logical blocks are available:

— drcCoefficientsParametricDrc()

— parametricDrcInstructions()

— parametricDrcTypeFeedForward()

— parametricDrcTypeLimiter()

Except for the drcCoefficientsParametricDrc() block, multiple instances of a logical block can appear in the bitstream. For the definition of a DRC set, a drcInstructionsUniDrc() block can refer to a DRC gain set defined in drcCoefficientsUniDrc() and/or to a DRC gain set defined in drcCoefficientsParametricDrc(). If a drcCoefficientsUniDrc() and a drcCoefficientsParametricDrc() block for the same *drcLocation* are present, the first *gainSetIndex* referring to drcCoefficientsParametricDrc() starts with an offset of the *gainSetCount* defined in drcCoefficientsUniDrc(). There is no distinction between parametric DRCs and DRCs based on dynamic gain values in the DRC selection process, because the selection is based on the drcInstructionsUniDrc() blocks. The specification in this subclause equally holds in the context of the extended payloads downmixInstructionsV1(), drcCoefficientsUniDrcV1(), drcInstructionsUniDrcV1() and loudnessInfoV1().

For parametric DRCs that require a make-up gain, the *gainOffset* field within drcInstructionsUniDrc() is employed.

Detailed information on each logical block can be found in 6.6.2. Algorithmic details are specified in 6.6.3.

6.6.2 Description of logical blocks

6.6.2.1 drcCoefficientsParametricDrc()

A drcCoefficientsParametricDrc() block includes top-level fields for the definition of parametric DRC gain sets. The *drcLocation* field is required for finding the first *gainSetIndex* referring to drcCoefficientsParametricDrc() (see Table Table A.53). The *parametricDrcFrameSize* field indicates the processing frame size. The maximum delay that can be incurred by any parametric DRC set in the stream can be specified in *bsParametricDelayMax.* Parametric DRCs can be reset at the decoder by using the *resetParametricDrc* field (e.g. in streaming scenarios). A reset shall initialize all internal algorithmic states to their default value. *parametricDrcGainSetCount* signals the number of independent parametric DRC configurations in the bitstream.

Each parametric DRC gain set refers to a parametricDrcInstructions() block by a unique *parametricDrcId*. *sideChainConfigType* defines the side-chain configuration of a parametric DRC gain set. For a value of 1, *levelEstimChannelWeightFormat* defines whether a simple channel map (*addChannel*[]) or individual channel weights (*channelWeight*[]) are provided for the definition of the side-chain input signal. For all other values, the side-chain input signal is defined by the *drcChannelGroup* the parametric DRC gain set is assigned to within a drcInstructionsUniDrc() block. If a drcInstructionsUniDrc() block with *downmixId*=X refers to a DRC gain set with present but non-matching *downmixId*=Y, *sideChainConfigType* shall be set to 0. For DRC sets with *drcSetEffect* "Duck other" and *sideChainConfigType* not equal to 1, the side-chain signal is defined by the *drcChannelGroup* the ducking gain set is associated with.

For normalization of the DRC side-chain input level, each parametric DRC gain set can define an individual *drcInputLoudness* value. Alternatively, it can be selected from an applicable loudnessInfo() structure. The applicable loudnessInfo() structure is selected according to 6.3 (Loudness normalization), wherein the requested *drcSetId* shall be zero (audio-signal without DRC processing) and the requested *downmixId* shall match the *downmixId* of the processed DRC set. Loudness values matching the processed *drcChannelGroup* (or to a customized side-chain input signal) shall be preferred if present (e.g. for MPEG-H 3D Audio).

6.6.2.2 parametricDrcInstructions()

The *parametricDrcId* field represents a unique identifier for each parametricDrcInstructions() block. Each parametric DRC can define an individual DRC look-ahead delay if required. For constant delay audio decoders, the definition of an application-specific maximum look-ahead delay (*parametricDrcLookAheadMax*) for parametric DRCs is required. For such applications, it is recommended to include the *bsParamtericDrcDelayMax* parameter in the bitstream that specifies a constant decoder delay that is large enough to cover the lookahead delay of any parametric DRC set in the stream. During the decoding process, a delay shall be applied to compensate for any difference between *parametricDrcLookAheadMax* and *parametricDrcLookAhead*. For *parametricDrcLookAhead* > *parametricDrcLookAheadMax*, the DRC tool shall set *parametricDrcLookAhead* = *parametricDrcLookAheadMax*. For simultaneously applied DRC sets, *parametricDrcLookAheadMax* should account for all parametric DRC instances in the chain.

The *parametricDrcPresetId* index can be used to efficiently select a suitable *parametricDrcType* and corresponding parametric DRC settings (see Table A.62)

If *parametricDrcPresetId* is not present, *parametricDrcType* indicates the type of parametric DRC that shall be used. Dependent on the selected type, further payloads follow in the bitstream that allow configuring specific parameters of a DRC algorithm:

— *parametricDrcType*==0x0: Parameters defined by parametricDrcTypeFeedForward() block (see 6.6.3.1 for algorithmic details).

— *parametricDrcType*==0x1: Parameters defined by parametricDrcTypeLimiter() block (see 6.6.3.2 for algorithmic details).

The parametricDrcInstructions() block can include future extension payloads by the definition of new *parametricDrcType* values. In the event that a *parametricDrcType* value is received that is not supported by a decoder implementation, the DRC tool parser shall read and discard the bits (*otherBit*) of the unknown *parametricDrcType* payload. The corresponding parametric DRC instance shall be disabled in that case.

6.6.2.3 parametricDrcTypeFeedForward()

The parametricDrcTypeFeedForward() block represents parameters of a standard feed-forward DRC design. The *levelEstimKWeightingType* field indicates which filters shall be applied before level estimation. *levelEstimIntegrationTime* controls the integration time used for level estimation in multiples of *parametricDrcFrameSize*. The gain-mapping curve of the parametric DRC can be either set by a *drcCharacteristic* index according to ISO/IEC 23091-3 or by definition of individual curve segments. Parameters such as maximum-boost or maximum-compress can be realized within the DRC curve parameterization. Various gain-smoothing parameters can be customized if required. See 6.6.3.1 for further algorithmic details. The presence of a peak limiter is strongly recommended for this parametric DRC design since it does not include means for generation of clipping prevention gains for the intended target loudness. A parametric DRC instance of type *parametricDrcType*=0x0 can also be combined with a parametric DRC instance of type *parametricDrcType*=0x1 by using the *dependsOnDrcSet* field (see D.2.8).

6.6.2.4 parametricDrcTypeLimiter()

The parametricDrcTypeLimiter() block represents parameters of a time domain peak limiter which is designed to prevent clipping of the time domain output signal. The *parametricLimThreshold* field indicates the limiting threshold which is applied after loudness normalization. The *parametricLimAttackTime* and *parametricLimReleaseTime* parameters control the gain smoothing behaviour of the limiting algorithm. *parametricLimAttackTime* is implicitly provided by the *parametricDrcLookAhead* field in the parametricDrcInstructions() block. *parametricLimReleaseTime* can be explicitly conveyed by using the *bsParametricLimReleaseTime* field. See 6.6.3.2 for further algorithmic details. The parameters of parametricDrcTypeLimiter() shall not have any influence on the DRC pre-selection #9 according to 6.3.2.2.3. Vice versa, the *limiterPeakTarget* parameter does not control this peak limiter. Moreover, gain manipulation as outlined in 6.4.6 is not permitted for parametricDrcType=0x1.

6.6.3 Algorithmic details

6.6.3.1 Parametric DRC of type PARAM_DRC_TYPE_FF

The parametric DRC of type PARAM_DRC_TYPE_FF (0x0) is a single-band feed-forward design. A block diagram is depicted in Figure 9.

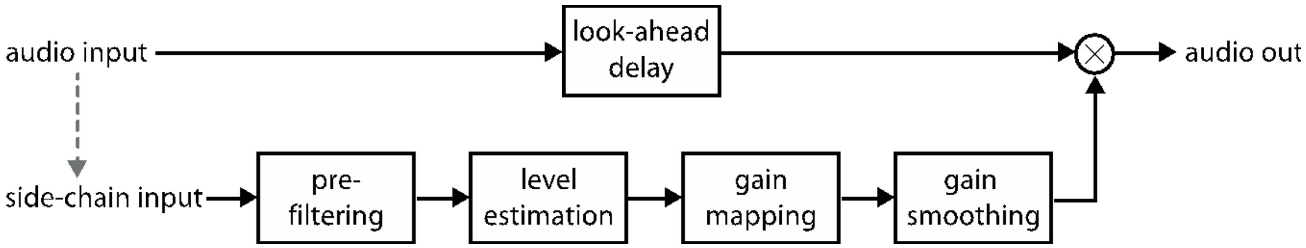


Figure 9 — Single-band feed-forward design of *parametricDrcType*==0x0

Dependent on *levelEstimKWeightingType*, the side-chain signal is filtered by two independent pre-filters according to ITU-R BS.1770-4. Each filter can be implemented by a second order section with the following transfer function:

$$H(z)=\frac{b_0+b_1z^{-1}+b_2z^{-2}}{a_0+a_1z^{-1}+a_2z^{-2}}$$

The filter coefficients according to ITU-R BS.1770-4 hold for a sampling rate of 48 kHz. Audio decoders that operate at other sampling rates will require different coefficient values, which should be chosen to provide the same frequency response that the specified filters provide at 48 kHz. For audio decoders that operate in the sub-band domain, the filters shall be realized by appropriate sub-band weighting. *parametricDrcFrameSize* shall be a multiple of the sub-band sampling interval. If the processing domain is not deterministic at the encoder, *parametricDrcFrameSize* shall be configured based on the sub-band sampling interval.

After pre-filtering, the level is estimated either in the time domain or sub-band domain according to Table 32. The energy of multiple parametric DRC processing frames (*parametricDrcFrameSize*) has to be accumulated dependent on *levelEstimIntegrationTime*.

Table 32 — Level estimation for time domain and sub-band domain processing for one parametric DRC frame (*parametricDrcType*==0x0)

```
/* level estimation for time domain processing */
estimLevelTimeDomain
{
    drcInputLoudnessTarget = -31;
    /* accumulate energy + normalize */
    levelLin = 0;
    for(c=0; c<nChannels; c++) {
        if (levelEstimChannelWeightFormat==0) {
            levelEstimChannelWeight[c] = addChannel[c];
        } else {
            levelEstimChannelWeight[c] = pow(10, 0.05 * channelWeight[c]);
        }
        for(t=0; t<levelEstimIntegrationTime; t++) {
            levelLin += levelEstimChannelWeight[c] * pow(audioSampleIn[c][t],2);
        }
    }
    levelLin = levelLin / levelEstimIntegrationTime;
        /* compute level */
    if (levelLin < 1e-10) levelLin = 1e-10;
```

```
    if (levelEstimKWeightingType == 2) {                // pre-filter ON
        levelDb = -0.691 + 10*log10(levelLin) + 3;
    } else {
        levelDb = 10*log10(levelLin) + 3;
    }
    levelDb = levelDb - drcInputLoudness + drcInputLoudnessTarget;
    return levelDb;
}
/* level estimation for sub-band domain processing. */
estimLevelSubbandDomain
{
    drcInputLoudnessTarget = -31;
    /* accumulate energy + normalize */
    /* levelEstimIntegrationTimeSb is the number of sub-band samples that fit into levelEstimIntegrationTime    */
    levelLin = 0;
    for(c=0; c<nChannels; c++) {
        if (levelEstimChannelWeightFormat==0) {
            levelEstimChannelWeight[c] = addChannel[c];
        } else {
            levelEstimChannelWeight[c] = pow(10, 0.05 * channelWeight[c]);
        }
        for(s=0; s<nDecoderSubbands; s++) {
            for(m=0; m<levelEstimIntegrationTimeSb; m++) {
                levelLin += levelEstimChannelWeight[c] * pow(abs(audioSampleSbInWeighted[c][s][m]),2);
            }
        }
    }
    levelLin = levelLin / (nDecoderSubbands * levelEstimIntegrationTimeSb);
    /* compute level */
    if (levelLin < 1e-10) levelLin = 1e-10;
    if (levelEstimKWeightingType == 2) {          // pre-filter ON
        levelDb = -0.691 + 10*log10(levelLin) + 3;
    } else {
        levelDb = 10*log10(levelLin) + 3;
    }
    levelDb = levelDb - drcInputLoudness + drcInputLoudnessTarget;
    return levelDb;
}
```


Next, the estimated level is mapped to a gain value according to Table 33, which also includes an initialization routine based on the received curve segments. An illustration of the curve parameterization for 5 curve nodes is depicted in Figure A.1.

Table 33 — Gain mapping for one parametric DRC frame (*parametricDrcType*==0x0)

```
mapLevelToGain
{
        /* initialization (only first frame) */
    initNodeLevel();
        /* determination of segment */
    for(c=0; c<nodeCount; c++) {
        if (levelDb <= nodeLevel[c]) {
            break;
        }
    }
        /* gain mapping */
    if (c == 0) {
        gainDb = nodeGain[c];
    } else if (c == nodeCount) {
        gainDb = nodeGain[c-1] - levelDb + nodeLevel[c-1];
    } else {
        gainDb = nodeGain[c] + (levelDb – nodeLevel[c]) /
            (nodeLevel[c-1] – nodeLevel[c]) * (nodeGain[c-1] – nodeGain[c]);
    }
    return gainDb;
}

initNodeLevel
{
        /* initialization (only for drcCurveDefinitionType==1) */
    for(c=0; c<nodeCount; c++) {
        if (c == 0) {
            nodeLevel[c] = nodeLevelInitial;
        }
        else {
            nodeLevel[c] = nodeLevel[c-1] + nodeLevelDelta[c];
        }
    }
    return nodeLevel;
}
```


Next, temporal smoothing is applied to the gains of consecutive parametric DRC frames. A pseudo-function of the gain smoothing is listed in Table 34.

Table 34 — Gain smoothing for one parametric DRC frame (*parametricDrcType*==0x0)

```
smoothGain(gainDb, gainSmoothDbPrevious, levelDb, levelSmoothDbPrevious)
{
        /* initialization (only first frame) */
    initSmoothingParameters();
        /* get alpha */
    levelDelta = levelDb-levelSmoothDbPrevious;
    if (gainDb < gainSmoothDbPrevious) {
```

```
            /* attack */
        if (levelDelta > gainSmoothAttackThreshold) {
            alpha = alphaAttackFast;
        } else {
            alpha = alphaAttackSlow;
        }
    } else {
            /* release */
        if (levelDelta < -gainSmoothReleaseThreshold) {
            alpha = alphaReleaseFast;
        } else {
            alpha = alphaReleaseSlow;
        }
    }
            /* smooth gain and level */
    if (gainDb < gainSmoothDbPrevious || holdCounter == 0) {
        levelSmoothDb = (1-alpha) * levelSmoothDbPrevious + alpha * levelDb;
        gainSmoothDb = (1-alpha) * gainSmoothDbPrevious + alpha * gainDb;
    }
            /* holdCounter */
    if (holdCounter > 0) {
        holdCounter = holdCounter – 1;
    }
    if (gainDb < gainSmoothDb) {
        holdCounter = holdOffCount;
    }
    return (gainSmoothDb,levelSmoothDb);
}

initSmoothingParameters
{
    alphaAttackFast    = 1 - exp(-1.0 * parametricDrcFrameSize / (gainSmoothAttackTimeFast * fs * 0.001));
    alphaReleaseFast = 1 - exp(-1.0 * parametricDrcFrameSize / ( (gainSmoothReleaseTimeFast * fs * 0.001));
    alphaAttackSlow    = 1 - exp(-1.0 * parametricDrcFrameSize / (gainSmoothAttackTimeSlow * fs * 0.001));
    alphaReleaseSlow = 1 - exp(-1.0 * parametricDrcFrameSize / (gainSmoothReleaseTimeSlow * fs * 0.001));
    holdOffCount = floor(gainSmoothHoldOff * 0.0053 * fs / parametricDrcFrameSize);
    levelSmoothDbPrevious = -135;
    gainSmoothDbPrevious = 0;
    holdCounter = 0;
}
```


Finally, the smoothed gains are interpolated and applied to the audio signal in the time domain or sub-band domain according to Table 35. Gain modifications as outlined in 6.4.6 can be equally applied to parametric DRC gain sets that were generated with *parametricDrcType*=0x0.

parametricDrcLookAhead indicates the delay that shall be applied to the audio signal before the computed gains are applied. The delay can be used to compensate for delay that is introduced to the computed gains by level estimation (*levelEstimIntegrationTime*), gain smoothing (e.g. *gainSmoothAttackTimeFast*) and linear interpolation during gain application.

Table 35 — Gain application for time domain and sub-band domain processing for one DRC frame (*parametricDrcType*==0x0)

```
applyGainTimeDomain(audioSampleIn, gainSmoothDb, gainSmoothPrevious)
{
    if (parametricDrcLookAheadPresent == 1) {
        drcLookAheadSamples = parametricDrcLookAhead * 0.001 * fs;
    }
    else {
        drcLookAheadSamples = parametricDrcLookAheadDefault * 0.001 * fs;
    }
    delayAudio(audioFrame, drcLookAheadSamples);
    numDrcSubframes = drcFrameSize/parametricDrcFrameSize;
    for(dsf=0; dsf<numDrcSubframes; dsf++) {
        gainInDb = gainSmoothDb[dsf];
        gainInLin = toLinearParametricDrc(gainInDb);
        gainInDiff = gainInLin - gainSmoothPrevious;
        for (t=0; t<parametricDrcFrameSize; t++) {
            audioSampleOut[dsf*parametricDrcFrameSize + t] = audioSampleIn[dsf*parametricDrcFrameSize + t] *
                                    ( gainSmoothPrevious + (t + 1) / parametricDrcFrameSize * gainInDiff );
        }
        gainSmoothPrevious = gainInLin;
    }
    return (audioSampleOut, gainSmoothPrevious);
}

applyGainSubbandDomain(audioSampleSbIn, gainSmoothDb, gainSmoothPrevious) {
    if (parametricDrcLookAheadPresent == 1) {
        drcLookAheadSamples = parametricDrcLookAhead * 0.001 * fs;
    }
    else {
        drcLookAheadSamples = parametricDrcLookAheadDefault * 0.001 * fs;
    }
    L = drcFrameSize/drcFrameSizeSb; /* downsampling factor (hopsize) */
    delayAudioSb(audioFrameSb, floor(drcLookAheadSamples/L));
    cnt = 0;
    m = 0;
    numDrcSubframes = drcFrameSize/parametricDrcFrameSize;
    for(dsf=0; dsf<numDrcSubframes; dsf++) {
        gainInDb = gainSmoothDb[dsf];
        gainInLin = toLinearParametricDrc(gainInDb);
        gainInDiff = gainInLin - gainSmoothPrevious;
        for (t=0; t<parametricDrcFrameSize; t++) {
            if (cnt == L-1) {
                for (s=0; s<nDecoderSubbands; s++) {
                    audioSampleSbOut[s][m] = audioSampleSbIn[s][m] *
                                ( gainSmoothPrevious +    (t + 1) / parametricDrcFrameSize * gainInDiff );
                }
                m++;
                cnt = -1;
            }
```

```
                cnt++;
            }
            gainSmoothPrevious = gainInLin;
        }
        return (audioSampleSbOut, gainSmoothPrevious);
}

toLinearParametricDrc(gainDb) {
        EFFECT_BIT_CLIPPING = 0x0100;           /* drcSetEffect 9 (Clip.Prev.) */
        EFFECT_BIT_FADE         = 0x0200;           /* drcSetEffect 10 (Fade) */
        EFFECT_BITS_DUCKING = 0x0400 | 0x0800; /* drcSetEffect 11 or 12 (Ducking) */
        gainRatio = 1.0;
        if (((drcSetEffect & EFFECT_BITS_DUCKING) == 0) &&
                        (drcSetEffect != EFFECT_BIT_FADE) && (drcSetEffect != EFFECT_BIT_CLIPPING)) {
            if (gainDb < 0.0) {
                gainRatio *= compress;
            }
            else {
                gainRatio *= boost;
            }
        }
        if (gainScalingPresent) {
            if (gainDb < 0.0) {
                gainRatio *= attenuationScaling;
            }
            else {
                gainRatio *= amplificationScaling;
            }
        }
        if (duckingScalingPresent && (drcSetEffect & EFFECT_BITS_DUCKING)) {
            gainRatio *= duckingScaling;
        }
        gainLin = pow(2.0, gainRatio * gainDb / 6.0);
        if (gainOffsetPresent) {
            gainLin *= pow(2.0, gainOffset / 6.0);
        }
        return (gainLin);
}
```


6.6.3.2 Parametric DRC of type PARAM_DRC_TYPE_LIM

The parametric DRC of type PARAM_DRC_TYPE_LIM (0x1) is a time domain peak limiter design, which is fully specified in Annex G. For audio decoders that operate in the sub-band domain, PARAM_DRC_TYPE_LIM should be only configured for application at the end of the signal path where time domain processing is possible. A block diagram is depicted in Figure 10.

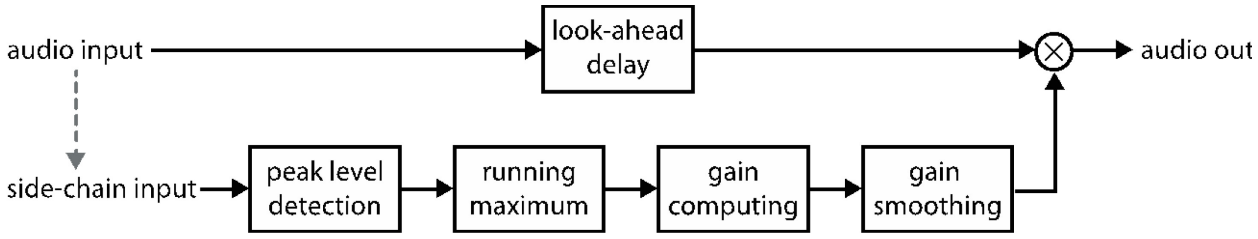


Figure 10 — Time domain peak limiter design of *parametricDrcType*==0x1

If not explicitly conveyed in the bitstream, the limiting algorithm will use the following default parameters (see also Table A.78):

— *parametricLimThreshold* = -1 dBFS

— *parametricLimAttackTime* = 5 ms

— *parametricLimReleaseTime* = 50 ms

6.6.4 Combining parametric and non-parametric DRCs

In the most generic case, parametric and non-parametric DRCs can be simultaneously active. The order of DRC processing is specified in 6.3.5. Since any parametric DRC can introduce delay, the audio signal and DRC gains shall be synchronized as specified below.

The DRC gains of non-parametric DRCs are synchronized with the non-delayed audio signal after EQ. If these gains are applied after one or more parametric DRCs, the gains shall be delayed by the same delay as the audio signal at the location where the gains are applied. The audio delay in a stage is as large as the maximum parametric DRC delay in this stage across all channels.

Figure 11 shows the generic delay structure and topology for mixed DRC processing including EQ for one channel. The figure shows all possible DRC locations in the processing topology, but only a maximum of three DRCs can be active and there can only be either a parametric DRC or a non-parametric DRC in a stage. For constant delay processing, the pre-downmix delay d_1 and the post-downmix delay d_2 shall not change when switching DRCs or EQs. If provided in the bitstream, the values of *parametricDrcDelayMax* and *eqDelayMax* shall be sufficiently large to achieve overall constant delay based on the constant pre- and post-downmix delays d_1 and d_2 when DRCs or EQs are applied. If constant delay operation is not active, the additional delays may be zero and the delays d_1 and d_2 may depend on the specific configuration.

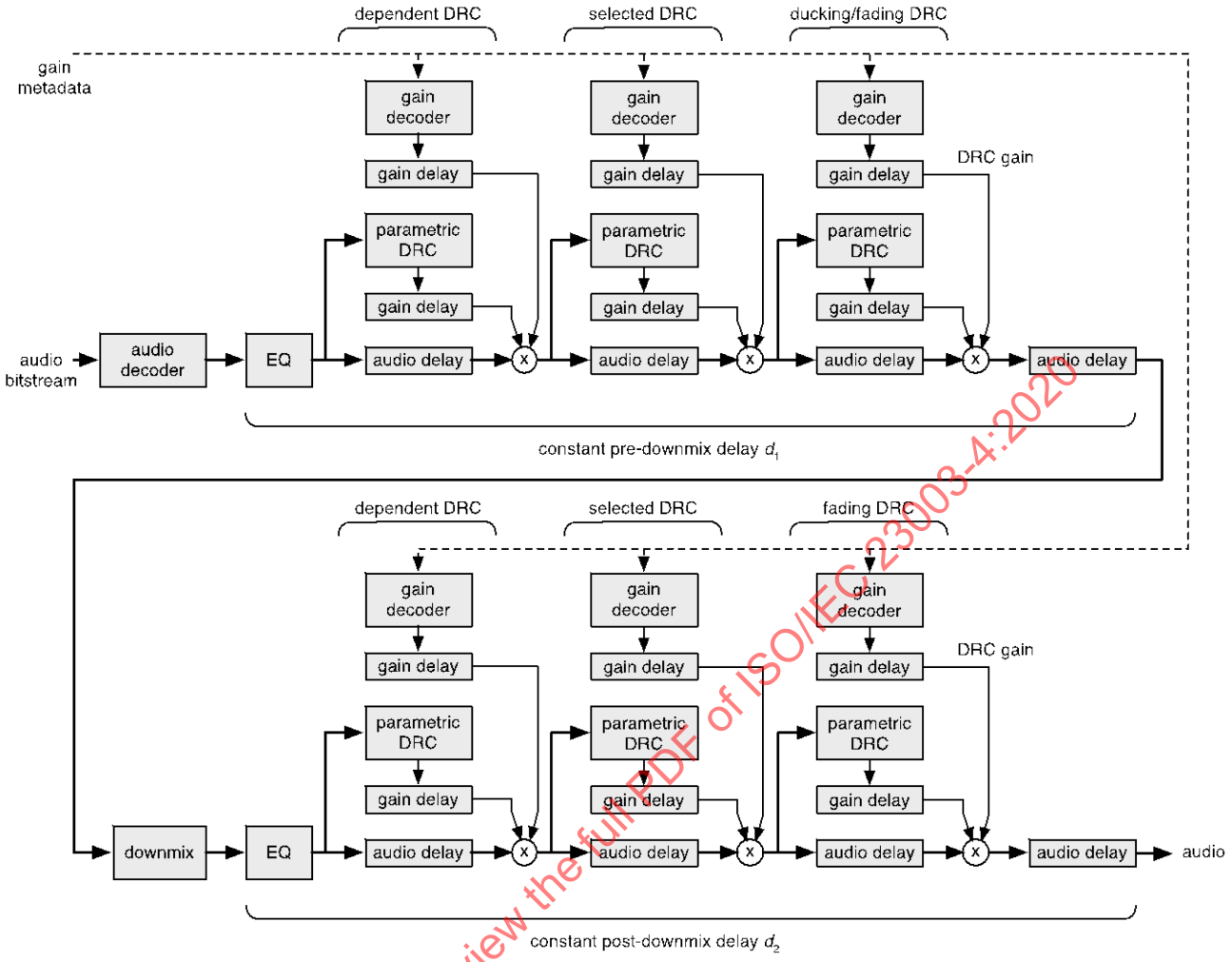


Figure 11 — Generic delay structure and processing topology for one audio channel

6.7 Loudness equalization support

The loudEqInstructions() payload provides support for loudness equalization, also known as loudness compensation, which aims to compensate the effect of the playback level on the tonal balance. The loudEqInstructions() repurpose DRC gain sequences to convey the dynamic acoustic level of spectral ranges. Although not described here because it is out of scope, these levels can for instance be used to control dynamic loudness equalization filters.

The *loudEqSetId* is a unique ID for each loudness equalization (LEQ) set. Each LEQ set declares to which combinations of *downmixId* and *drcSetId* it can be applied. Not more than one LEQ set can be declared for a given combination of *downmixId* and *drcSetId*. Each LEQ set declares one or more DRC gain sequences it depends on. For each of the gain sequences, the DRC characteristic that was used at the encoder side, the frequency range that should be equalized based on the sequence, a scaling factor, and offset are declared. The time-varying loudness information to control LEQ filters may be derived by applying the inverse DRC characteristic and using the loudness information of the corresponding loudnessInfo() payload. If the inverse characteristic is applied before the interpolation, linear interpolation should be used, i.e. the slope information should be ignored for sequences represented in spline format. The scaling factor and offset are applied in this order after the inverse DRC characteristic in the dB domain. An example LEQ system is discussed in D.2.10.

6.8 Equalization tool

6.8.1 Overview

The equalization (EQ) tool provides EQ filters that are applied to EQ channel groups. The set of EQ filters applied to all EQ channel groups is called "EQ set". EQ can be applied in the time domain or the sub-band domain, for instance, in the QMF domain of an audio decoder. A time-domain EQ filter for a channel group is called a "filter cascade" because it is composed of a cascade of filter blocks, where each block contains one or more filter elements that work in parallel as shown in Figure 12.

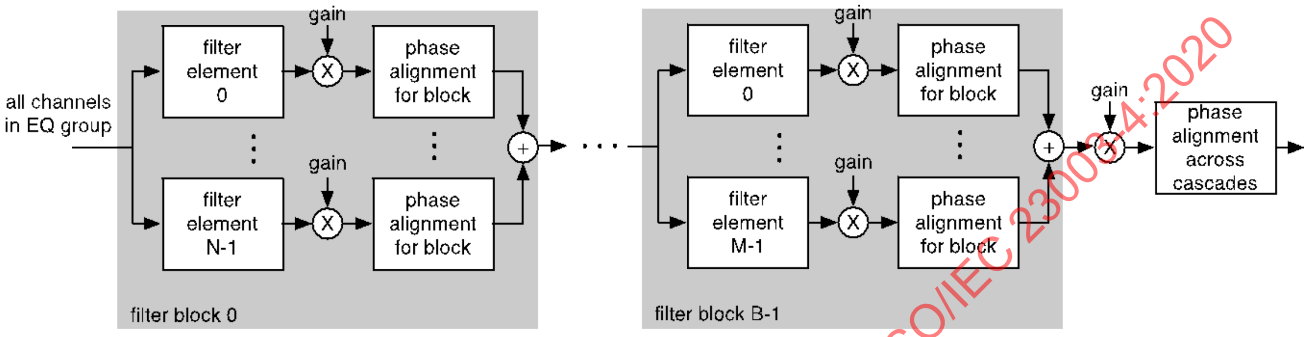


Figure 12 — Generic EQ filter cascade structure

EQ can be applied before and/or after a downmix. A bitstream field controls whether phase alignment is applied. If the field is set, the EQ filter cascades are phase aligned across EQ channel groups to avoid undesired phase effects. In the sub-band domain, the EQ effect is obtained by applying sub-band gains to channel groups of the audio signal.

If EQ is applied together with DRC, the DRC gain sequences shall be time aligned with the equalized audio signal.

The EQ parameters are conveyed by the payloads, eqCoefficients() and eqInstructions() as described in the following. Annex H provides additional considerations regarding the equalization tool.

6.8.2 EQ payloads

6.8.2.1 Overview

The EQ payloads are transmitted in the uniDrcConfigExtension() payload. They consist of two parts:

— eqCoefficients() including the EQ filter parameters;

— eqInstructions(), each including an EQ set to achieve a certain effect.

The loudness information for EQ processed audio is reflected in the loudnessInfoV1() payload.

6.8.2.2 eqCoefficients() payload

The eqCoefficients() payload in Table 86 defines all available filter elements and filter blocks. Filter elements can be conveyed in three different formats:

— Pole/zero

— FIR coefficients

— Sub-band gains

The two time-domain formats, pole/zero and FIR coefficients, are mutually exclusive, i.e. only one of them can be used. Sub-band gains can be specified per band or using a spline representation. The different filter elements are indexed. The filter blocks are composed of one ore more filter elements referred to by an index and an optional filter element gain factor. The available filter blocks can be referenced by the eqInstructions() payload by index.

6.8.2.3 eqInstructions() payload

The eqInstructions() payload in Table 89 contains metadata to generate a complete EQ filter set for all EQ channel groups. For each channel group, a filter cascade is composed of one or more filter blocks that are defined in the eqCoefficients() payload and a cascade gain factor. Phase alignment flags indicate whether phase alignment is active for two or more channel groups.

The EQ set is externally referred to by an ID specified in the *eqSetId* field. The *downmixId* and *additionalDownmixIds* indicate which downmixes the EQ set can be applied to. The *eqApplyToDownmix* flag controls whether the EQ is applied before or after a downmix. A second EQ set may be required, depending on the *dependsOnEqSet* field. The second EQ set shall be located at the opposite side of the downmix block. The EQ set shall not be applied without a second EQ set if the *noIndependtEqUse* flag is set. One of multiple DRC sets specified by an ID shall be applied. A *drcSetId* of 0 indicates that applying no DRC is a permitted option.

If the *dependsOnEqSet* field in an eqInstructions() payload has a value of 1, the auxillary information of the payload describes the combined EQ including the dependent one. For the combined EQ, the auxillary information of the dependent eqInstructions() is ignored. The auxillary information includes (*additional*)*downmixId*, (*additional*)*drcSetId*, and *eqPurpose*.

The *eqTransitionDuration* indicates the duration of the crossfade between the output of the previous EQ set and the current EQ set when the EQ changes mid-stream. The default value for *eqTransitionDuration* is 0.05 s.

6.8.2.4 LoudnessInfo payload for EQ

The loudnessInfoV1() payload is identical with the loudnessInfo() payload except for the additional *eqSetId* field that indicates which EQ set is active. The loudnessInfoV1() payload is transmitted in the loudnessInfoSetExtension() payload.

6.8.3 EQ filter elements

6.8.3.1 Supported filter elements in pole/zero format

A filter element has the following generic pole/zero form in the complex *z*-domain:

$$H_{\text{element,PZ}}(z) = g_{\text{element}} \frac{\prod_{m-1}^{M}\left(1-c_m z^{-1}\right)}{\prod_{k=1}^{K/2}\left(1-d_k z^{-1}\right)^2}$$

The gain coefficient g_{element} is a real value. The class of filters that can be used as a filter element has several constraints.

Constraints for all the poles, assuming a pole has a radius, r, and angle, θ ($d_k = re^{j\theta}$):

— K is even or zero.

— All poles are inside the unit circle (|r|<1).

— For each pole, there is a second pole at the same location.

— If a complex pole is present, the conjugate complex pole shall be present as well (if $\theta \neq n\pi : re^{j\theta}, re^{-j\theta}$ are present).

Constraints for all the zeros, assuming a zero has a radius, r, and angle, ϑ ($c_m = re^{j\vartheta}$):

— M is even or zero.

— An even number of real zeros with |r|==1 can be present.

— Each real zero with $|r| \neq 1$ is part of a pair of zeros at $r, \frac{1}{r}$.

— Each generic zero is part of a quadruple of zeros at $re^{j\theta}, re^{-j\theta}, \frac{1}{r}e^{j\theta}, \frac{1}{r}e^{-j\theta}$.

The constraints for the poles ensure that the filter is stable and that an all-pass filter with the same phase response can be created. The constraints for the zeros ensure that the corresponding FIR filter has linear phase and a constant integer delay when measured in units of a sample interval. Further restrictions according to Table 37 apply to filter elements for which an all-pass filter shall be created to achieve phase alignment.

6.8.3.2 Supported filter elements in FIR coefficient format

In general, a filter element in FIR coefficient format can be any symmetrical FIR filter (symmetric or anti-symmetric). Taking advantage of the symmetry, only the first half of the coefficients is transmitted. An FIR filter can only be used if it is not considered in any phase alignment as described in 6.8.5.1.2. Therefore, it can only be used in filter blocks that have only a single filter element and if the filter block is not part of a filter cascade with phase alignment.

6.8.3.3 Supported filter elements in sub-band gain format

Sub-band gains represent the spectral shape of the EQ filter. The gains are applied to the sub-band signals, typically in the QMF domain of an audio decoder.

6.8.4 EQ set selection

Conceptually, the EQ set selection has lower priority than the DRC set selection. Therefore, the DRC set selection is processed first and if there are still different EQ sets during the final selection step, the EQ set is selected, which is appropriate for the requested EQ purpose. If there is no EQ request from the host, a default EQ is requested. DRCs that are automatically applied, such as for ducking or fading, are not affected by the EQ selection. A mid-stream change of the EQ payload does not trigger the selection process, i.e. the selected EQ set ID continues to be valid but an EQ crossfade may be needed if the corresponding EQ filter parameters have changed.

The selected EQ set may have a dependent EQ set that shall be applied as well.

6.8.5 Application of EQ set

The selected EQ sets are applied to EQ channel groups of the audio signal before and/or after the downmix. Figure 13 shows an example processing chain with EQ and DRC. If a DRC set is applied at the

same location before or after the downmix, the EQ set shall be applied before the DRC set. The channel grouping for EQ and DRC is independent.

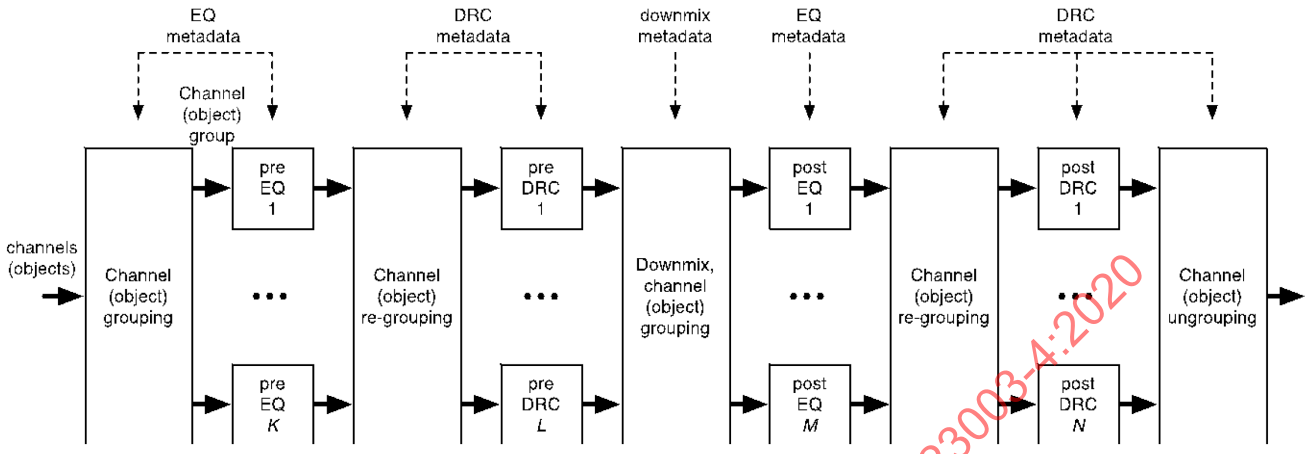


Figure 13 — Example overview of processing chain with EQ, DRC, and downmix

If the EQ payload changes or a different EQ is selected mid-stream, the outputs of the previous EQ processing and current EQ processing are cross-faded for the specified duration according to the *bsTransitionDuration* field using a linear cross-fade characteristic.

6.8.5.1 Time domain EQ

6.8.5.1.1 Decoding of filter element parameters

6.8.5.1.1.1 Decoding of *z*-domain poles and zeros

Due to the filter element constraints, only a fraction of the actual pole and zero locations need to be transmitted. The remaining ones are recovered by the decoder according to Table 36 and Table 37. Therefore, it is sufficient to transmit zeros and poles in the range of $r = [0,1]$ and $\theta = [0,\pi]$ only.

Table 36 — Transmitted and decoded pole locations ($n \in \{0,1\}$)

Transmitted pole; associated bitstream count field	Decoded poles
Real pole: $re^{j\theta}$, ($r<1$, $\theta == n\pi$); *realPoleCount*	Real double pole: $re^{j\theta}$, $re^{j\theta}$
Complex pole : $re^{j\theta}$, ($r<1$, $\theta \neq n\pi$); *complexPoleCount*	Double conjugate pole pair: $re^{j\theta}$, $re^{j\theta}$, $re^{-j\theta}$, $re^{-j\theta}$

Table 37 — Transmitted and decoded zero locations ($n \in \{0,1\}$)

Transmitted zero; associated bitstream count field	Decoded zero(s)
Generic zero: $re^{j\theta}$, (any r, any θ); *genericZeroCount*	Two conjugate zero pairs: $re^{j\theta}$, $re^{-j\theta}$, $\frac{1}{r}e^{j\theta}$, $\frac{1}{r}e^{-j\theta}$
Real zero (for restricted use)[a]: $re^{j\theta}$, ($r == 1$, $\theta == n\pi$); *realZeroRadiusOneCount*	Real zero: $re^{j\theta}$

Real zero[b]: $re^{j\theta}$, ($r<1$, $\theta == n\pi$); *realZeroCount*	Inverse zero pair: $re^{j\theta}$, $\frac{1}{r}e^{j\theta}$

[a] Only permitted if used in filter blocks that contain only a single filter element and if these filter blocks are not subject to cascade phase alignment.

[b] For each real zero with $\theta == 0$, multiply the linear filter element gain factor by -1 to compensate for the phase inversion.

The radius, r, of a pole or zero is decoded according to Table A.95 and by computing the radius from ρ using $r = 1 - \rho$. The ρ values of Table A.95 can be generated using the result for *r_quant_offs* of the pseudo code in Table 38.

Table 38 — Pseudo code to generate the zero/pole radius decoding table

```
r_quant_step_min = 0.2;
for (n=0; n<128; n++) {
    tmp[n]=1.0/(0.025*(127-n));
}
for (n=0; n<128; n++) {
    r_quant_step[n] = tmp[n] + r_quant_step_min – tmp[1];
}
sumlevel[0] = r_quant_step[0];
for (n=1; n<128; n++) {
    sumlevel[n] = sumlevel[n-1] + r_quant_step[n];
}
for (n=0; n<128; n++) {
    r_quant[n] = 1 - pow(10.0, -0.05*sumlevel[n]);
    r_quant_offs[n] = pow(10.0, -0.05*sumlevel[n]);
}
```

The angle θ of a pole or zero is decoded according to Table A.96 and by computing the angle θ by $\theta = \alpha\pi$. The table can be generated by the pseudo code in Table 39 using the result for *angle_quant*. For real poles or zeros, the angle is determined by the associated sign field. If the sign is 1, the angle is $\theta = \pi$, otherwise, $\theta = 0$.

Table 39 — Pseudo code to generate the pole/zero angle decoding table

```
for (n=0; n<12; n++) {              /* 12 steps per octave */
    octave[n] = pow(2.0, -n/12.0);
}
for (k=0; k<11; k++) {
    for (n=0; n<12; n++) {
        tmp[k*12 + n] = octave[n] * pow(2.0, -k);
    }
}
tmp[127] = 0.0;
for (i=0; i<127; i++) {
    angle_quant = tmp[127-i];
}
```

Once all poles and zeros are decoded, a corresponding filter is created. It is recommended that each pair of poles and pair of zeros be grouped and implemented using a second-order section. The remaining zeros are implemented by an FIR filter. In detail, the recommended steps are:

a) The pole that is closest to the unit circle should be paired with the zero that is closest to it in the z-plane where the radius of zeros with $r > 1$ is replaced by the inverse radius $1/r$ for the distance calculation.

b) Rule (1) should be repeatedly applied until all the poles have been paired with zeros.

c) The resulting second-order sections should be ordered according to decreasing closeness of poles to the unit circle.

d) The remaining zeros are processed using an FIR filter.

6.8.5.1.1.2 Decoding of FIR coefficients

The FIR coefficients are decoded according to Table A.93. Since only the first half of the coefficients is transmitted, the second half is generated according to Table 40.

Table 40 — Pseudo code to generate the FIR filter coefficients

```
for (i=0; i<firFilterOrder/2+1; i++) {
    firCoef[i] = decodeFirCoef(bsFirCoefficient[i]);
}
for (i=0; i<(firFilterOrder+1)/2; i++) {
    if (firSymmetry==1) {
        firCoef[firFilterOrder-i] = - firCoef[i];
    } else {
        firCoef[firFilterOrder-i] = firCoef[i];
    }
}
if ((firSymmetry==1) && ((firFilterOrder & 1)==0)) {
    firCoef[firFilterOrder/2] = 0.0;
}
```

6.8.5.1.2 Phase alignment of time-domain filter

All-pass filters are employed to achieve phase alignment. Phase alignment is always necessary for all filter elements in each filter block. Phase alignment shall also be done at the output of filter cascades if the corresponding phase alignment flag is set.

For phase alignment on a filter block level, the phase response of each filter element in the block is matched by an all-pass filter. For filter elements in pole/zero format that have more poles than zeros ($K > M$), a delay of $P = (K - M)/2$ shall be added to the filter element so that phase matching can be achieved as described below. The matching all-pass filter is then inserted after all filter elements except the ones with the matching phase response.

The phase alignment at the filter cascade level is controlled by the phase alignment flags. For phase alignment at this level, the phase response of each cascade shall be matched by a corresponding cascade of filter blocks with all-pass filters that match the phase response of the corresponding filter elements. Each all-pass cascade shall then be inserted after all cascades except the one with the matching phase response.

Identical all-pass filters that appear in all parallel paths at the filter block or cascade filter level shall be eliminated. To minimize delay, the aggregated all-pass delays as indicated by L in all parallel paths shall be reduced by the same amount until at least one path has zero delay. Similarly, the delay added to filter elements with $K > M$ shall be reduced as much as possible.

The all-pass with matching phase response of a filter element is derived from the poles and zeros of a filter element in pole/zero format as follows. In mathematical terms, the transfer function of the all-pass filter with a magnitude response of unity can be written as:

$$H_{\text{allpass,PZ}}(z) = \prod_{k=1}^{k/2}\left(-d_k^*\right)\frac{\prod_{k=1}^{K/2}\left(1 - z^{-1}/d_k^*\right)}{\prod_{k=1}^{K/2}\left(1 - d_k z^{-1}\right)} z^{-L}$$

The phase response of a basic EQ filter is matched by the all-pass filter when:

- Each single pole of the all-pass filter corresponds to a pair of poles of the filter element. Both poles are at the same location.
- The location of the zeros of the all-pass filter is given by the inverse conjugate complex pole location of the all-pass filter.
- The delay of L = (M– K)/2 samples is necessary if the filter element has more zeros than poles (M > K).

If the filter element has FIR coefficient format, phase alignment is not supported for that element.

6.8.5.2 Sub-band domain EQ

6.8.5.2.1 Decoding of sub-band gains

Sub-band gains shall be provided that cover the entire audio spectrum. Depending on the sub-band representation of the rendering system, the sub-band gains may need to be interpolated and resampled using the applicable sub-band center frequencies. A number of specific sub-band formats are supported, for instance, some common QMF configurations. The sub-band gain values shall be present for each

band in ascending (frequency) order or the spectral gain function shall be given by a spline representation as indicated by the *eqSubbandGainRepresentation* field.

If *eqSubbandGainRepresentation* == 0, the sub-band gains are decoded according to Table A.91. They are applied to the sub-bands of the audio decoder, usually the QMF domain starting with the first gain for the sub-band with the lowest center frequency.

If *eqSubbandGainRepresentation* == 1, the sub-band EQ gains are represented in a spline format. Table 41 contains pseudo code for the spline interpolation of the gains (Table 42 contains specifications of parameters and functions used in Table 41). The interpolation is done in the dB domain and the gain values are converted to linear values before they are applied to the sub-band signals. The spline nodes are located on a sparse grid with approximately a third octave frequency resolution. The sub-band EQ gains are calculated by evaluating the spline at the center frequency of each sub-band.

Table 41 — Pseudo code for decoding and spline interpolation to produce an EQ sub-band gain vector gEqSubband[b] for the b-th sub-band

```
eqNodeCountMax = 33;
eqNodeIndexMax = eqNodeCountMax - 1;
sEqNode[0] = decodeEqSlope(eqSlopeCode[0]);
gEqNode[0] = decodeEqInitialGain(eqGainInitialCode);
eqNodeFreqIndex[0] = 0;
eqNodeFreq[0] = decodeEqNodeFreq(eqNodeFreqIndex[0]);
for (n=1; n<nEqNodes; n++) {
    sEqNode[n] = decodeEqSlope(eqSlopeCode[n]);
    gEqNode[n] = gEqNode[n-1] + decodeEqGainDelta(eqGainDeltaCode[n]);
    fEqDelta[n] = decodeEqFreqDelta(eqFreqDeltaCode[n]);
    eqNodeFreqIndex[n] = eqNodeFreqIndex[n-1] + fEqDelta[n];
    eqNodeFreq[n] = decodeEqNodeFreq(eqNodeFreqIndex[n]);
}
if ((eqNodeFreq[nEqNodes-1] < audioSampleFreq/2.0) &&
                (eqNodeFreqIndex[nEqNodes-1] < eqNodeIndexMax)) {
    sEqNode[nEqNodes] = 0;
    gEqNode[nEqNodes] = gEqNode[nEqNodes-1];
    fEqDelta[nEqNodes] = eqNodeIndexMax - eqNodeFreqIndex[nEqNodes-1];
    eqNodeFreqIndex[nEqNodes] = eqNodeIndexMax;
    eqNodeFreq[nEqNodes] = decodeEqNodeFreq(eqNodeFreqIndex[nEqNodes]);
    nEqNodes += 1;
}
for (n=0; n<nEqNodes-1; n++) {
    for (b=0; b<eqSubbandGainCount; b++) {
        fSub = max(fCenter[b], eqNodeFreq[0]);
        fSub = min(fSub, eqNodeFreq[nEqNodes-1]);
        if ((fSub >= eqNodeFreq[n]) && (fSub <= eqNodeFreq[n+1])) {
            warpedDeltaFreq = warpFreqDelta (fSub, eqNodeFreqIndex[n]);
            gEqSubbandDb[b] = interpolateEqGain(fEqDelta[n+1], gEqNode[n], gEqNode[n+1],
                                sEqNode[n], sEqNode[n+1], warpedDeltaFreq);
            gEqSubband[b] = pow(2.0, gEqSubbandDb[b] / 6.0);
        }
    }
}
```

Table 42 — Specifications of parameters and functions used in Table 41

Parameter/function	Description
audioSampleFreq	Sample rate of audio signal in Hz.
fCenter[b]	Center frequency of *b*-th sub-band in Hz.

fLo = 20;
fHi = 24000;
eqNodeCountMax = 33;
stepRatio = (log(fHi) / log(fLo) - 1) / (eqNodeCountMax -1);
	Common parameters.

decodeEqNodeFreq(eqNodeFreqIndex) {
  return(pow(fLo, 1 + eqNodeFreqIndex * stepRatio));
}	

warpFreqDelta(fSubband, eqNodeFreqIndex) {
  return((log(fSubband)/log(nodeFrequency[0]) – 1) /
  stepRatio – eqNodeFreqIndex);
}	

interpolateEqGain()	See Table 43.
decodeEqSlope()	See Table A.98.
decodeEqInitialGain()	See Table A.100.
decodeEqGainDelta()	See Table A.101.
decodeEqFreqDelta()	See Table A.99.

Table 43 — Interpolation of sub-band EQ gains for one spline segment

```
interpolateEqGain(bandStep, gain0, gain1, slope0, slope1, f)
{
    float k1, k2, a, b, c, d;
    float nodesPerOctaveCount = 3.128;
    float gainLeft = gain0;
    float gainRight = gain1;
    float slopeLeft = slope0 / nodesPerOctaveCount;
    float slopeRight = slope1 / nodesPerOctaveCount;
    float bandStepInv = 1.0 / (float)bandStep;
    float bandStepInv2 = bandStepInv * bandStepInv;
    k1 = (gainRight - gainLeft) * bandStepInv2;
    k2 = slopeRight + slopeLeft;
    a = bandStepInv * (bandStepInv * k2 - 2.0 * k1);
    b = 3.0 * k1 - bandStepInv * (k2 + slopeLeft);
    c = slopeLeft;
    d = gainLeft;
    result = (((a * f + b ) * f + c ) * f ) + d;
    return result;
}
```

If sub-band gains are not available but the EQ set is applied to the sub-band domain, the sub-band gains are computed from the time-domain filter elements at the sub-band center frequencies $\omega = \omega_c$ (*bandIndex*) in radians.

The magnitude response of a single zero is:

$$\left|H_{\text{zero},m}\left(e^{j\omega}\right)\right| = \left[1 + r^2 - 2r\cos\left(\omega - \theta\right)\right]^{0.5}$$

The magnitude response of a single pole is:

$$\left|H_{\text{pole},k}\left(e^{j\omega}\right)\right| = \left[1 + r^2 - 2r\cos\left(\omega - \theta\right)\right]^{-0.5}$$

Hence, the magnitude of a filter element in pole/zero format is determined by:

$$\left|H_{\text{element, PZ}}\left(e^{j\omega}\right)\right| = g_{\text{element}} \prod_{m=1}^{M}\left|H_{\text{zero},m}\left(e^{j\omega}\right)\right| \prod_{k=1}^{K}\left|H_{\text{pole},k}\left(e^{j\omega}\right)\right|$$

For the FIR coefficient format, the magnitude response of the filter element can be computed as shown in Table 44, assuming that the impulse response is denoted by $h(n)$.

Table 44 — Magnitude response of FIR filter

FIR type	Order *M*	Symmetry	Magnitude response $\lvert H_{element}(e^{j\Omega})\rvert$	Coefficients
1	Even	Even	$g_{\text{element}} \sum_{m=0}^{M/2} \alpha(m)\cos(m\omega)$	$a(0) = h\left(\frac{M}{2}\right)$ $\alpha(m) = 2h\left(\frac{M}{2} - m\right), \quad m \in \left[1, \frac{M}{2}\right]$
3	Even	Odd	$g_{\text{element}} \sum_{m=1}^{M/2} \alpha(m)\sin(m\omega)$	
2	Odd	Even	$g_{\text{element}} \sum_{m=1}^{(M+1)/2} \beta(m)\cos\left(\left(m - \frac{1}{2}\right)\omega\right)$	$\beta(m) = 2h\left(\frac{(M+1)}{2} - m\right)$, $m \in \left[1, \frac{M+1}{2}\right]$
4	Odd	Odd	$g_{\text{element}} \sum_{m=1}^{(M+1)/2} \beta(m)\sin\left(\left(m - \frac{1}{2}\right)\omega\right)$	

The magnitude response of a filter cascade with B filter blocks where each block has $E(b)$ filter elements is therefore:

$$\left|H_{\text{cascade}}\left(e^{j\omega}\right)\right| = g_{\text{cascade}} \prod_{b=1}^{B}\left[\sum_{e=1}^{E(b)}\left|H_{\text{element},e,b}\left(e^{j\omega}\right)\right|\right]$$

The transmission of time-domain EQ filters only is sufficient to also cover sub-band EQ. However, if only sub-band gains are transmitted without specifying the time-domain EQ filters, Table 45 should be used to determine the applicable EQ. Time-domain filters shall be specified if the EQ is to be applied in the time-domain.

Table 45 — Conversion of EQ filter formats

Transmitted EQ formats	Domain where EQ is applied	Conversion
Time domain and sub-band domain	Sub-band	Compute sub-band gains from time-domain filters if EQ sub-band resolution does not match and sub-band gains are not available in spline format.
Time domain	Sub-band	Compute sub-band gains from time-domain filters.
Sub-band	Sub-band	If the EQ gains are not given in spline format, resample using linear interpolation in dB domain, if necessary.
Sub-band	Time-domain	EQ cannot be applied.

6.9 Complexity management

6.9.1 General

A DRC tool decoder implementation should determine or include a value $L_{C,\mathrm{tot,max}}$ for the absolute total complexity level permitted for processing. Based on that value, the DRC tool does not permit configurations that exceed the complexity limit based on the complexity estimation. Therefore, a configuration can only be applied if the following condition is met:

$$\log_2(C_{\mathrm{DRC,tot}} + C_{\mathrm{EQ,tot}}) \leq L_{C,\mathrm{tot,max}}$$

The definitions of $C_{\mathrm{DRC,tot}}$ and $C_{\mathrm{EQ,tot}}$ are given in the following subclauses. Application standards and audio coding standards that include this specification may require that $L_{C,\mathrm{tot,max}}$ exceeds a given minimum value.

It is recommended that the DRC tool decoder disables all DRC sets with $\log_2(C_{\mathrm{DRC,tot}}) > L_{C,\mathrm{tot,max}}$ and all EQ sets with $\log_2(C_{\mathrm{EQ,tot}}) > L_{C,\mathrm{tot,max}}$ before the DRC selection process. The combined complexity of DRC and EQ can be evaluated after the DRC selection process. If it exceeds the permitted complexity, the EQ should be disabled, unless the *requiresEQ* field of the DRC has a value of 1. If the complexity still exceeds the limit, the primary DRC should be disabled and the DRC selection process should be reiterated.

6.9.2 DRC and downmixing complexity estimation

The total complexity $C_{\mathrm{DRC,tot}}$ for DRC processing and downmixing is estimated in the DRC tool decoder based on the *drcSetComplexityLevel* field value $L_{C,\mathrm{DRC}}$ in the drcInstructionsUniDrcV1() payloads and the downmix matrix, if applied.

$$C_{\mathrm{DRC,tot}} = \frac{f_{\mathrm{s}}}{f_{\mathrm{s,norm}}}\left(\sum_{m=1}^{3} N_{ch,m}\, 2^{L_{C,\mathrm{DRC},m}} + \sum_{i=1}^{N_{\mathrm{Base}}}\sum_{j=1}^{N_{\mathrm{Target}}} D_{i,j}\right)$$

The parameters are defined as:

f_{s} output audio sample rate in Hz.

$f_{\mathrm{s,norm}}$ constant of 48 000 Hz.

$L_{C,\mathrm{DRC},1}$ represents the complexity level of a primary DRC, if present. Otherwise, it is 0.

$L_{C,\mathrm{DRC},2}$ represents the complexity level of a dependent DRC, if present. Otherwise, it is 0.

$L_{C,\mathrm{DRC},3}$ represents the complexity level of a ducking or fading DRC, if present. Otherwise, it is 0.

$N_{ch,m}$ number of audio channels at the location of DRC_m, if applicable. Otherwise, it is 0.

N_{Base} audio channel count of base layout.

N_{Target} audio channel count after downmix.

$D_{i,j}$ has a value of 0 if no downmix is applied or if the corresponding downmix coefficient has a value of 0; otherwise, it is 1 for time-domain processing or 2 for sub-band processing.

The DRC tool encoder computes the complexity level for each DRC set as follows:

$$L_{C,\mathrm{DRC}} = \max(0,\mathrm{ceil}(L'_{C,\mathrm{DRC}}))$$

with

$$L'_{C,\mathrm{DRC}} = \log_2\left(\frac{1}{I}\left(\sum_{i=1}^{I}\left(w_{\mathrm{Mod}}P_i + w_{IIR}Q_i + w_{Lap}R_i + w_{Shape}S_i\right) + \sum_{k=1}^{K}T_k\right)\right)$$

DRC sets with $L'_{C,\mathrm{DRC}} > 15$ are not permitted. The parameters are defined as:

I audio channel count at the DRC location. If the corresponding drcInstructionsUniDrcV1() payload includes a *downmixId* of 0×7F, it is the channel count of the base layout.

w_{Mod} weighting factor that considers the complexity of applying the DRC gain to the audio signal. It has a value of 1.0 for time-domain processing or 2.0 for sub-band processing.

P_i has a value of 1 if a DRC set is applied to channel i. Otherwise, it is 0.

w_{IIR} weighting factor of 5.0 to consider the IIR filter complexity.

Q_i number of pairs of a single zero and pole which occur in the Linkwitz-Riley filters for time-domain band splitting including all phase-alignment filters in audio channel i. The value is zero for a single band time-domain DRC or any sub-band domain DRC. For time-domain DRCs with more than four bands, the Linkwitz-Riley filter bank cannot be used. A decoder can support such DRCs by using a QMF analysis and synthesis bank instead. However, in that case, the complexity of the QMF processing is not included in the *drcSetComplexityLevel* field and Q_i has a value of 0.

w_{Lap} weighting factor of 2.0 for the complexity of the sub-band overlap processing.

R_i number of DRC bands for audio channel i, if a multiband DRC is applied in the sub-band domain and *drcBandType*==1. Otherwise, the value is 0.

w_{Shape} weighting factor of 6.0 for the complexity of the shaping filter processing.

S_i number of pairs of a single zero and pole that occur in the shaping filter bank for audio channel i. The value is 0 if no shaping filter is applied.

K number of different DRC sequences that are actively applied to the audio signal.

T_k complexity value for computing DRC gain values of each gain sequence k that is applied to the audio signal according to Table 46.

Table 46 — Complexity values for computing DRC gains

Gain sequence generation	DRC domain	*T*
Spline interpolation (*interpolationType* == 0)	Time domain	5.0
Linear interpolation (*interpolationType* == 1)	Time domain	2.5
Spline interpolation (*interpolationType* == 0)	Sub-band	0.0
Linear interpolation (*interpolationType* == 1)	Sub-band	0.0
Parametric DRC (*parametricDrcType*==0)	Time domain	$A(5B + 1) + 3$
Parametric DRC (*parametricDrcType*==0)	Sub-band	$5A$
Parametric limiter (*parametricDrcType*==1)	Time domain	$4.5A + 136\sqrt{C / D}$
Key: *A*: channel count at side chain input *B*: weighting filter order *C*: *parametricLimAttackTime* *D*: default value of *C*		

The complexity of a dependent DRC set is not included in the result. $L_{C,\mathrm{DRC}}$ is 0 if *bsSequenceIndex*==0. The *drcSetComplexityLevel* value applies to the domain where the DRC is processed by default given a fully bitstream compatible audio decoder.

To ensure that the DRC tool decoder implementation supports a certain complexity level, tests should include edge-case configurations that individually maximize the complexity contribution of each one of the parameters in the complexity estimation formula.

6.9.3 EQ complexity estimation

The total complexity $C_{\mathrm{EQ,tot}}$ for EQ processing is estimated in the decoder based on the *eqSetComplexityLevel* field value $L_{\mathrm{C,EQ}}$ in the eqInstructions() payloads:

$$C_{\mathrm{EQ,tot}} = \frac{f_{\mathrm{s}}}{f_{\mathrm{s,norm}}} \sum_{i=1}^{2} N_{ch,i}\, 2^{L_{C,\mathrm{EQ},i}}$$

The parameters are defined as:

f_{s} output audio sample rate in Hz

$f_{\mathrm{s,norm}}$ constant of 48 000 Hz

$L_{C,\mathrm{EQ},1}$ represents the complexity level of a primary EQ set, if present. Otherwise, it is 0.

$L_{C,EQ,2}$ represents the complexity level of a dependent EQ set, if present. Otherwise, it is 0.

$N_{ch,1}$, $N_{ch,2}$ represent the audio channel count at the location before and after the downmix, where the primary and dependent EQ sets are applied, if present. Otherwise, it is 0.

The DRC tool encoder computes the complexity level for each EQ set by:

$$L_{C,EQ} = \max(0,\text{ceil}(L'_{C,EQ}))$$

with

$$L'_{C,EQ} = \log_2\left(\frac{1}{I}\sum_{i=1}^{I}\left(w_{Sb}U_i + w_{IIR}V_i + w_{FIR}Z_i\right)\right)$$

EQ sets with $L'_{C,EQ} > 15$ are not permitted. The parameters are defined as:

I audio channel count at the EQ location. If the corresponding eqInstructions() payload includes a *downmixId* of 0x7F, it is the channel count of the base layout.

w_{Sb} weighting factor of 2.5 that considers the complexity of applying the EQ gain to the sub-band signal.

U_i has a value of 1 if the EQ is applied in the sub-band domain of channel i. Otherwise, it is 0.

w_{IIR} weighting factor of 5.0 that considers the IIR filter complexity.

V_i number of pairs of a single zero and pole that occur in IIR filters including all phase-alignment filters applied to channel i in the time domain, if present. Otherwise, it is 0.

w_{FIR} weighting factor of 0.4 that considers the FIR filter complexity.

Z_i total order of all FIR filters applied to channel i in the time domain, if present. Otherwise, it is 0.

$L'_{C,EQ}$ is 0 if the EQ instructions of both, *tdFilterCascadePresent* and *subbandGainsPresent* have a value of 0 and the audio signal is processed in the time-domain. The complexity of a dependent EQ set is not included in the result.

The *eqSetComplexityLevel* value applies to the domain where the EQ set is specified. If it is specified in both domains, it applies to the time domain and the sub-band domain EQ complexity is set to 2.5. If the EQ filters are specified in the time domain but applied in the sub-band domain, the EQ complexity value for the sub-band domain is also set to 2.5.

The EQ can include filter elements consisting of a combination of an IIR and FIR filter. To ensure that the DRC tool decoder implementation supports a certain complexity level, tests should include the time-domain edge-case configurations at the maximum supported complexity for EQ with IIR filters only (Z_i = 0) and EQ with FIR filters only (V_i = 0).

6.10 Loudness normalization

6.10.1 Overview

Loudness normalization is done by the DRC decoder if a target loudness level (*targetLoudness*) is supplied to the decoder. If no value is supplied, loudness normalization is disabled. If the loudness

normalization would result in a too high level (that causes clipping or severe audible distortions introduced by a peak limiter), the output level can be decreased by reducing the gain as described in 6.3.2. The DRC decoder accepts an optional value *loudnessDeviationMax* to limit the applied gain reduction. Annex F provides some general considerations for loudness normalization.

6.10.2 Loudness normalization based on target loudness

The target loudness is the desired output loudness level in LKFS (loudness, K-weighted[4]). If the target loudness value is supplied, a loudness normalization gain is applied to all channels of the time domain signal as the final step after all DRC related processing, if applicable.

The loudness normalization is based on the loudness metadata received in the applicable loudnessInfo() structure. The DRC decoder accepts an optional value *loudnessNormalizationGainDbMax* to limit the maximum loudness normalization gain. Second, it accepts an optional value *outputPeakLevelMax* to limit the peak level as described in 6.3.2. Third, it accepts an optional value *loudnessNormalizationGainModificationDb* to modifiy the default *loudnessNormalizationGainDb* based on external parameters or measurements available only at the decoder. *loudnessNormalizationGainModificationDb* can be different for each DRC frame.

The DRC decoder has an input for loudness normalization settings which accepts three indices to request a particular loudness measurement method, measurement system, and pre-processing as specified in Table 47, Table 48, and Table 49. If no settings are supplied, default values are used according to Table 50.

Table 47 — Permitted method definition indices for the loudness normalization settings

Index	Method definition
0	Default method
1	Program Loudness (as defined in loudnessInfo())
2	Anchor loudness (as defined in loudnessInfo())

Table 48 — Measurement system indices for the loudness normalization settings

Index	Measurement system
0	Default system
1	ITU-R BS.1770–4
2	User
3	Expert/Panel
4 *(reserved, not permitted)*	Reserved Measurement System A (RMS_A)
5 *(reserved, not permitted)*	Reserved Measurement System B (RMS_B)
6 *(reserved, not permitted)*	Reserved Measurement System C (RMS_C)
7 *(reserved, not permitted)*	Reserved Measurement System D (RMS_D)
8 *(reserved, not permitted)*	Reserved Measurement System E (RMS_E)

Table 49 — Permitted measurement system pre-processing indices for the loudness normalization settings

Index	Pre-processing
0	Default preprocessing
1	No preprocessing
2	High-pass filter with 500Hz cutoff frequency as specified in Table A.50

Table 50 — Default loudness normalization settings

Entry	Default value	Meaning of default
Measurement method	1	Program loudness
Measurement system	1	ITU-R BS.1770–4
Pre-processing	1	No pre-processing

Table 51 — Matching order for measurement system

Order of selection	Requested measurement system as defined in Table 48							
	BS.1770–4	User	Expert/ Panel	RMS_A	RMS_B	RMS_C	RMS_D	RMS_E
	Selected measurement system as defined in Table A.50							
1	BS.1770–4	User	Expert/ Panel	RMS_A	RMS_B	RMS_C	RMS_D	RMS_E
2	RMS_C	RMS_E	RMS_D	RMS_C	RMS_C	RMS_B	Expert/ Panel	User
3	RMS_B	RMS_D	RMS_C	RMS_B	RMS_A	RMS_A	RMS_C	RMS_D
4	RMS_A	Expert/ Panel	RMS_B	BS.1770–4	BS.1770–4	BS.1770–4	RMS_B	Expert/ Panel
5	RMS_D	RMS_C	RMS_A	RMS_D	RMS_D	RMS_D	RMS_A	RMS_C
6	Expert/ Panel	RMS_B	BS.1770–4	Expert/ Panel	Expert/ Panel	Expert/ Panel	BS.1770–4	RMS_B
7	RMS_E	RMS_A	RMS_E	RMS_E	RMS_E	RMS_E	RMS_E	RMS_A
8	User	BS.1770–4	User	User	User	User	User	BS.1770–4

The request is then fulfilled by the decoder with best effort depending on which loudness metadata values are available. This is accomplished by the following steps:

1) From the applicable loudnessInfo() structure, select all loudness measurements that match the requested measurement method. The applicable loudnessInfo() structure is defined by the selected *drcSetId* and the requested *downmixId* as specified in 6.3. If no match is found, select all loudness measurements that match the other method for the applicable loudnessInfo() structure. If no match is found, fallback values are selected from a different loudnessInfo() structure as specified in 6.1.3, first for the requested method. Second, the other method is used if no match was found. Finally, if

no match was found at all, disable loudness normalization and ignore the remaining steps. The same holds if no applicable loudnessInfo() structure is available.

2) Within all selected loudness measurements, find the one that matches the requested measurement system. If none is found, use the order given in Table 51 to find the best match. The value of *contentLoudness* is equal to the selected loudness value. All column entries for the requested measurement system shall be processed including the reserved measurement systems to ensure that measurement systems defined in the future can be found and applied. Although the reserved measurement systems are currently not permitted, the system shall be able to process requests of those systems.

3) If pre-processing is requested, the loudness value needs to be adjusted. The adjustment value ΔL_{pre} is the difference between the program loudness based on ITU-R BS.1770-4 with pre-processing and ITU-R BS.1770-4 without pre-processing. If the two loudness values are not available, the adjustment value is set to −2 dB. The value of *contentLoudness* is equal to the selected loudness value plus adjustment.

4. If pre-processing is requested and a value for *deviceCutOffFrequency* $f_{c,\text{pre}}$ is provided (see Table A.102), the value of *contentLoudness* is calculated as stated in step 3, but with a modified adjustment value ΔL_{pre}:

$$\Delta L_{\text{pre}} = \Delta L_{\text{pre}} \frac{\max[20, \min(500, f_{c,\text{pre}})] - 20}{500 - 20} \tag{5}$$

For each DRC frame, the loudness normalization is then applied according to Table 52.

Table 52 — Loudness normalization processing

```
if (targetLoudnessPresent) {
    loudnessNormalizationGainDb = targetLoudness – contentLoudness;
}
else {
    loudnessNormalizationGainDb = 0.0;
}
if (loudnessNormalizationGainDbMaxPresent) {
    loudnessNormalizationGainDb = min(loudnessNormalizationGainDb, loudnessNormalizationGainDbMax);
}
if (loudnessNormalizationGainModificationDbPresent) {
    gainNorm = pow(2.0, (loudnessNormalizationGainDb + loudnessNormalizationGainModificationDb) / 6);
} else {
    gainNorm = pow(2.0, loudnessNormalizationGainDb / 6);
}
for (t=0; t<drcFrameSize; t++) {
    for (c=0; c<nChannels; c++) {
        audioSample[c][t] = gainNorm * audioSample[c][t];
    }
}
```

6.11 DRC in streaming scenarios

6.11.1 DRC configuration

The DRC configuration information is usually conveyed by the ISO base media file format (ISO/IEC 14496-12) syntax. However, if a legacy streaming format such as ADTS is used to carry an MPEG audio stream that does not support the ISO base media file format, the configuration information needs to be embedded in the audio stream. This can be achieved by sending uniDrcConfig() as part of the uniDrc() payload in-stream. Whenever uniDrcConfig() is transmitted, then a loudnessInfoSet() shall be transmitted as well. Since the configuration information is only required at a lower rate than the DRC frame rate, presence flags are used to indicate when this information is available. Each received loudnessInfoSet() and uniDrcConfig() payload shall be decoded and processed. DRC frames with configuration information should be synchronized with random access points of the employed streaming format.

The full DRC information can only be decoded after loudnessInfoSet() and uniDrcConfig() were received by the decoder. The repetition rate of uniDrcConfig() determines the maximum decoding delay for a decoder that starts decoding the content at an arbitrary position of the stream. In such a scenario when the decoder receives uniDrc() but no uniDrcConfig(), the decoder shall mute the audio output until uniDrcConfig() is available and the DRC information can be successfully decoded and applied to the audio signal. However, if the uniDrcConfig() is not received within 2.5 s, the decoder shall disable any DRC- and loudness-related processing and un-mute the audio.

If the DRC configuration information is both conveyed by the ISO base media file format syntax and by the uniDrcConfig() syntax, the in-stream syntax shall take precedence over the ISO base media file format syntax.

If loudness information is both conveyed by the ISO base media file format syntax and by the in-stream loudnessInfoSet() syntax, the loudness information with file format syntax shall take precedence over the in-stream loudness information. Once loudness information for a track is received in the ISO base media file format syntax, subsequently occurring loudness information with loudnessInfoSet() syntax within samples of that track shall be ignored.

6.11.2 Error handling

If uniDrc() payloads are corrupted or lost on the receiver side during DRC gain application, the DRC decoder shall replace missing DRC frames by artificial uniDrc() payloads with *drcGainCodingMode* = 0 ("simple" mode). If the last valid DRC gain was an attenuation gain, *gainInitialCode* shall be set to the last valid DRC gain. If the last valid DRC gain was an amplification gain, *gainInitialCode* shall be set to 0dB. The same applies for all DRC channel groups and DRC bands. The described behaviour shall be repeated till the next valid uniDrc() payload is received. However, if a valid uniDrc() payload is not received within 2.5 s, the decoder shall disable any DRC-and loudness-related processing until a valid uniDrc() payload is received.

6.12 DRC configuration changes during active processing

After the DRC tool decoder is initialized and has started to process audio frames, there are several events that can trigger a reconfiguration or an adjustment of parameters as summarized in Table 53 when certain values of the payload change with respect to the previous values.

Table 53 — Configuration changes for various events

Payload received	Condition	Result
loudnessInfoSet(), no uniDrcConfig()	If different values in loudnessInfoSet().	New selection of DRC set based on latest request. Reset of DRC tool if selected DRC set changes. Update loudness normalization gain.
loudnessInfoSet(), uniDrcConfig()	If same values in all fields of uniDrcConfig() but different values in loudnessInfoSet().	New selection of DRC set based on latest request. Reset of DRC tool if selected DRC set changes. Update loudness normalization gain.
	If different value in any field of uniDrcConfig().	Reset of DRC tool. New selection of DRC set based on latest request.
uniDrcInterface()	If different value in any field ignoring *compress*, *boost*, and *loudnessNormalizationGainModificationDb.*	New selection of DRC set based on latest request. Reset of DRC tool if selected DRC set changes.
	If different values in *compress*, *boost*, or *loudnessNormalizationGainModificationDb* fields but same values in all remaining fields.	Update loudness normalization gain, *compress*, and *boost* values.

A DRC tool decoder reset is associated with an initialization that discards the current state except for the latest request in the form of a uniDrcInterface() payload (see Annex B) or equivalent and the last DRC gains and loudness normalization values that were applied. When the decoder is initialized after the reset, it uses the latest DRC request to select a DRC set. Discontinuities or instant level changes of the output signal shall be avoided by using methods such as smoothing or cross-fading. For instance, linear interpolation can be applied to smoothly transition from the previous gain sequence to the new one after reset. The interpolation starts at the minimum gain node of the received DRC frame when the reset occurs and ends at the minimum gain node of the next frame. Similarly, the loudness normalization gain is linearly interpolated during the first frame after reset. If the DRC set is applied in a location before or after downmix where no DRC set was applied before reset, the gain value to interpolate from is set to 0 dB. Vice versa, if a DRC set is not used anymore in a location where a DRC set has been used before, the gain value to interpolate to in the second frame is 0 dB.

If any of the active DRC sets during a DRC set change includes a time-domain multi-band DRC, no gain interpolation is applied. Instead, a fade-out can be applied to the audio signal before the reset and a fade-in after the reset. The multi-band filter states are initialized with zero.

If the configuration change includes a target channel layout change or a target channel count change, no interpolation is applied from previous DRC gains nor from the previous normalization gain. The reset does not disrupt the output audio signal.

An alternative recommended method to achieve a smooth transition is by cross-fading the output of the previous configuration with the output of the new configuration before discarding the old configuration, which requires that two instances of the processor with different configurations are running in parallel when a transition is in progress. This method can be universally applied independently of the type of configuration change except if the number of output channels changes.

The processing strategy for dynamic DRC set changes may slightly differ as long as discontinuities are prevented and smooth transitions are limited to a maximum duration of four DRC frames.

An update of the loudness normalization gain, *loudnessNormalizationGainModificationDb*, *compress*, or *boost* value does not involve a reset. The *compress* and *boost* values are smoothly adjusted to the new

values by linear interpolation during one DRC frame. A loudness normalization gain adjustment in one DRC frame cannot be faster than ± 40 dB/s with linear interpolation. If more frames are needed to reach the new gain value, a rate of ± 40 dB/s is applied until the target value is reached. The loudness normalization gain is limited to a maximum value so that *outputPeakLevelMax* is not exceeded (see 6.3.2.2.3).

In the ISO base media file format framework (see ISO/IEC 14496-12), the same rules apply. For instance, if a LoudnessBox is received, the loudness normalization gain is updated and configuration changes can trigger a reset as indicated in Table 53.

7 Syntax

7.1 Syntax of DRC payload

The syntax of uniDrc() payload is shown in Table 54.

Table 54 — Syntax of uniDrc() payload

Syntax	No. of bits	Mnemonic
uniDrc()		
{		
uniDrcLoudnessInfoSetPresent;	**1**	**bslbf**
if (uniDrcLoudnessInfoSetPresent==1) {		
uniDrcConfigPresent;	**1**	**bslbf**
if (uniDrcConfigPresent==1) {		
uniDrcConfig();		
}		
loudnessInfoSet();		
}		
uniDrcGain();		
}		

7.2 Syntax of DRC gain payload

The syntax of uniDrcGain() in-stream payload and for ISO/IEC 14496-12 is shown in Table 55. Table 56 shows the syntax of uniDrcGainExtension() payload.

Table 55 — Syntax of uniDrcGain() in-stream payload and for ISO/IEC 14496-12

Syntax	No. of bits	Mnemonic
uniDrcGain()		
{		
nDrcGainSequences = gainSequenceCount; /* from drcCoefficientsUniDrc or drcCoefficientsUniDrc V1 */		
for (s=0; s<nDrcGainSequences; s++) {		
if (gainCodingProfile[s]<3) {		

Syntax	No. of bits	Mnemonic
drcGainSequence();		
}		
}		
uniDrcGainExtPresent;	**1**	**bslbf**
if (uniDrcGainExtPresent==1) {		
uniDrcGainExtension();		
}		
}		

Table 56 — Syntax of uniDrcGainExtension() payload

Syntax	No. of bits	Mnemonic
uniDrcGainExtension()		
{		
while (**uniDrcGainExtType** !=UNIDRCGAINEXT_TERM) {	**4**	**uimsbf**
extSizeBits = **bitSizeLen** + 4;	**3**	**uimsbf**
extBitSize = **bitSize** + 1;	**extSizeBits**	**uimsbf**
switch (uniDrcGainExtType) {		
/* add future extensions here */		
default:		
for (i=0; i<extBitSize; i++) {		
otherBit;	**1**	**bslbf**
}		
}		
}		
}		

7.3 Syntax of static DRC payload

Tables 57 - 76 show the syntax of various forms of static DRC payload.

Table 57 — Syntax of uniDrcConfig() payload

Syntax	No. of bits	Mnemonic
uniDrcConfig()		
{		
sampleRatePresent;	**1**	**bslbf**
if (sampleRatePresent==1) {		
bsSampleRate;	**18**	**uimsbf**
}		
downmixInstructionsCount;	**7**	**uimsbf**

Syntax	No. of bits	Mnemonic
drcDescriptionBasicPresent;	**1**	**bslbf**
if (drcDescriptionBasicPresent==1) {		
drcCoefficientsBasicCount;	**3**	**uimsbf**
drcInstructionsBasicCount;	**4**	**uimsbf**
} else {		
drcCoefficientsBasicCount = 0;		
drcInstructionsBasicCount = 0;		
}		
drcCoefficientsUniDrcCount;	**3**	**uimsbf**
drcInstructionsUniDrcCount;	**6**	**uimsbf**
channelLayout();		
for (i=0; i<downmixInstructionsCount; i++) {		
downmixInstructions();		
}		
for (i=0; i<drcCoefficientsBasicCount ; i++) {		
drcCoefficientsBasic();		
}		
for (i=0; i<drcInstructionsBasicCount ; i++) {		
drcInstructionsBasic();		
}		
for (i=0; i<drcCoefficientsUniDrcCount; i++) {		
drcCoefficientsUniDrc();		
}		
for (i=0; i<drcInstructionsUniDrcCount; i++) {		
drcInstructionsUniDrc();		
}		
uniDrcConfigExtPresent;	**1**	**bslbf**
if (uniDrcConfigExtPresent==1) {		
uniDrcConfigExtension();		
}		
}		

Table 58 — Syntax of loudnessInfoSet() payload

Syntax	No. of bits	Mnemonic
loudnessInfoSet()		
{		
loudnessInfoAlbumCount;	**6**	**uimsbf**
loudnessInfoCount;	**6**	**uimsbf**

Syntax	No. of bits	Mnemonic
for (i=0; i<loudnessInfoAlbumCount; i++) {		
loudnessInfo();		
}		
for (i=0; i<loudnessInfoCount; i++) {		
loudnessInfo();		
}		
loudnessInfoSetExtPresent;	**1**	**bslbf**
if (loudnessInfoSetExtPresent==1) {		
loudnessInfoSetExtension();		
}		
}		

Table 59 — Syntax of loudnessInfo() payload

Syntax	No. of bits	Mnemonic
loudnessInfo()		
{		
drcSetId;	**6**	**uimsbf**
downmixId;	**7**	**uimsbf**
samplePeakLevelPresent;	**1**	**bslbf**
if (samplePeakLevelPresent==1) {		
bsSamplePeakLevel;	**12**	**uimsbf**
}		
truePeakLevelPresent;	**1**	**bslbf**
if (truePeakLevelPresent==1) {		
bsTruePeakLevel;	**12**	**uimsbf**
measurementSystem;	**4**	**uimsbf**
reliability;	**2**	**uimsbf**
}		
measurementCount;	**4**	**uimsbf**
for (i=0; i<measurementCount; i++) {		
methodDefinition;	**4**	**uimsbf**
methodValue;	**2..8**	**vlclbf**
measurementSystem;	**4**	**uimsbf**
reliability;	**2**	**uimsbf**
}		
}		

Table 60 — Syntax of loudnessInfoV1() payload

Syntax	No. of bits	Mnemonic
loudnessInfoV1()		
{		
drcSetId;	**6**	**uimsbf**
eqSetId;	**6**	**uimsbf**
downmixId;	**7**	**uimsbf**
samplePeakLevelPresent;	**1**	**bslbf**
if (samplePeakLevelPresent==1) {		
bsSamplePeakLevel;	**12**	**uimsbf**
}		
truePeakLevelPresent;	**1**	**bslbf**
if (truePeakLevelPresent==1) {		
bsTruePeakLevel;	**12**	**uimsbf**
measurementSystem;	**4**	**uimsbf**
reliability;	**2**	**uimsbf**
}		
measurementCount;	**4**	**uimsbf**
for (i=0; i<measurementCount; i++) {		
methodDefinition;	**4**	**uimsbf**
methodValue;	**2..8**	**vlclbf**
measurementSystem;	**4**	**uimsbf**
reliability;	**2**	**uimsbf**
}		
}		

Table 61 — Syntax of loudnessInfoSetExtension() payload

Syntax	No. of bits	Mnemonic
loudnessInfoSetExtension()		
{		
while (**loudnessInfoSetExtType** != UNIDRCLOUDEXT_TERM) {	**4**	**uimsbf**
extSizeBits = **bitSizeLen** + 4;	**4**	**uimsbf**
extBitSize = **bitSize** + 1;	**extSizeBits**	**uimsbf**
switch (loudnessInfoSetExtType) {		
UNIDRCLOUDEXT_EQ:		
loudnessInfoV1AlbumCount;	**6**	**uimsbf**
loudnessInfoV1Count;	**6**	**uimsbf**
for (i=0; i<loudnessInfoV1AlbumCount; i++) {		
loudnessInfoV1();		

Syntax	No. of bits	Mnemonic
}		
for (i=0; i<loudnessInfoV1Count; i++) {		
loudnessInfoV1();		
}		
break;		
/* add future extensions here */		
default:		
for (i=0; i<extBitSize; i++) {		
otherBit;	**1**	**bslbf**
}		
}		
}		
}		

Table 62 — Syntax of channelLayout() payload

Syntax	No. of bits	Mnemonic
channelLayout()		
{		
baseChannelCount;	**7**	**uimsbf**
layoutSignallingPresent;	**1**	**bslbf**
if (layoutSignallingPresent==1) {		
definedLayout;	**8**	**uimsbf**
if (definedLayout==0) {		
for (i=0; i<baseChannelCount; i++) {		
speakerPosition;	**7**	**uimsbf**
}		
}		
}		
}		

Table 63 — Syntax of downmixInstructions() payload

Syntax	No. of bits	Mnemonic
downmixInstructions()		
{		
downmixId;	**7**	**uimsbf**
targetChannelCount;	**7**	**uimsbf**
targetLayout;	**8**	**uimsbf**

Syntax	No. of bits	Mnemonic
downmixCoefficientsPresent;	**1**	**bslbf**
if (downmixCoefficientsPresent==1)		
for (i=0; i<targetChannelCount; i++) {		
for (j=0; j<baseChannelCount; j++) {		
bsDownmixCoefficient;	**4**	**bslbf**
}		
}		
}		
}		

Table 64 — Syntax of downmixInstructionsV1() payload

Syntax	No. of bits	Mnemonic
downmixInstructionsV1()		
{		
downmixId;	**7**	**uimsbf**
targetChannelCount;	**7**	**uimsbf**
targetLayout;	**8**	**uimsbf**
downmixCoefficientsPresent;	**1**	**bslbf**
if (downmixCoefficientsPresent==1)		
bsDownmixOffset;	**4**	**uimsbf**
for (i=0; i<targetChannelCount; i++) {		
for (j=0; j<baseChannelCount; j++) {		
bsDownmixCoefficientV1;	**5**	**bslbf**
}		
}		
}		
}		

Table 65 — Syntax of in-stream drcCoefficientsBasic() payload

Syntax	No. of bits	Mnemonic
drcCoefficientsBasic()		
{		
drcLocation;	**4**	**uimsbf**
drcCharacteristic;	**7**	**uimsbf**
}		

Table 66 — Syntax of drcCoefficientsBasic() payload for ISO/IEC 14496-12

```
aligned(8) class DRCCoefficientsBasic extends FullBox('udc1', version, flags=0) {
                    // N copies of this box, one of these per DRC_location
    bit(4)              reserved = 0;
    signed int(5)       DRC_location;
    unsigned int(7) DRC_characteristic;
}
```

Table 67 — Syntax of in-stream drcCoefficientsUniDrc() payload

Syntax	No. of bits	Mnemonic
drcCoefficientsUniDrc()		
{		
drcLocation;	**4**	**uimsbf**
drcFrameSizePresent;	**1**	**bslbf**
if (drcFrameSizePresent==1) {		
bsDrcFrameSize;	**15**	**uimsbf**
}		
gainSetCount;	**6**	**uimsbf**
for (i=0; i<gainSetCount; i++) {		
gainSetIndex=1;		
gainCodingProfile;	**2**	**uimsbf**
gainInterpolationType;	**1**	**uimsbf**
fullFrame;	**1**	**uimsbf**
timeAlignment;	**1**	**uimsbf**
timeDeltaMinPresent;	**1**	**bslbf**
if (timeDeltaMinPresent==1) {		
bsTimeDeltaMin;	**11**	**uimsbf**
}		
if (gainCodingProfile==3) {		
bandCount=1;		
} else {		
bandCount;	**4**	**uimsbf**
if (bandCount>1) {		
drcBandType;	**1**	**uimsbf**
}		
for (j=0; j<bandCount; j++) {		
drcCharacteristic;	**7**	**uimsbf**
}		
for (j=1; j<bandCount; j++) {		

Syntax	No. of bits	Mnemonic
if (drcBandType==1) {		
crossoverFreqIndex;	**4**	**uimsbf**
} else {		
startSubBandIndex;	**10**	**uimsbf**
}		
}		
}		
k=k+bandCount;		
}		
gainSequenceCount=k;		
}		

Table 68 — Syntax of in-stream drcCoefficientsUniDrcV1() payload

Syntax	No. of bits	Mnemonic
drcCoefficientsUniDrcV1()		
{		
drcLocation;	**4**	**uimsbf**
drcFrameSizePresent;	**1**	**bslbf**
if (drcFrameSizePresent==1) {		
bsDrcFrameSize;	**15**	**uimsbf**
}		
drcCharacteristicLeftPresent	**1**	**bslbf**
if (drcCharacteristicLeftPresent == 1) {		
characteristicLeftCount;	**4**	**uimsbf**
for (k=1; k<=characteristicLeftCount; k++) {		
characteristicLeftIndex = k;		
characteristicFormat;	**1**	**bslbf**
if (characteristicFormat==0) {		
bsGainLeft;	**6**	**uimsbf**
bsIoRatioLeft;	**4**	**uimsbf**
bsExpLeft;	**4**	**uimsbf**
flipSignLeft;	**1**	**bslbf**
} else {		
characteristicNodeCount = **bsCharNodeCount**+1;	**2**	**uimsbf**
for (n=1; n<=characteristicNodeCount; n++) {		
bsNodeLevelDelta;	**5**	**uimsbf**
bsNodeGain;	**8**	**uimsbf**
}		

Syntax	No. of bits	Mnemonic
}		
}		
}		
drcCharacteristicRightPresent	**1**	**bslbf**
if (drcCharacteristicRightPresent == 1) {		
characteristicRightCount;	**4**	**uimsbf**
for (k=1; k<=characteristicRightCount; k++) {		
characteristicRightIndex = k;		
characteristicFormat;	**1**	**bslbf**
if (characteristicFormat==0) {		
bsGainRight;	**6**	**uimsbf**
bsIoRatioRight;	**4**	**uimsbf**
bsExpRight;	**4**	**uimsbf**
flipSignRight;	**1**	**bslbf**
} else {		
characteristicNodeCount = **bsCharNodeCount**+1;	**2**	**uimsbf**
for (n=1; n<=characteristicNodeCount; n++) {		
bsNodeLevelDelta;	**5**	**uimsbf**
bsNodeGain;	**8**	**uimsbf**
}		
}		
}		
}		
shapeFiltersPresent;	**1**	**bslbf**
if (shapeFiltersPresent == 1) {		
shapeFilterCount;	**4**	**uimsbf**
for (k=1; k<=shapeFilterCount; k++) {		
shapeFilterIndex = k;		
lfCutFilterPresent;	**1**	**bslbf**
if (lfCutFilterPresent == 1) {		
lfCornerFreqIndex;	**3**	**uimsbf**
lfFilterStrengthIndex;	**2**	**uimsbf**
}		
lfBoostFilterPresent;	**1**	**bslbf**
if (lfBoostFilterPresent == 1) {		
lfCornerFreqIndex;	**3**	**uimsbf**
lfFilterStrengthIndex;	**2**	**uimsbf**
}		

Syntax	No. of bits	Mnemonic
hfCutFilterPresent;	**1**	**bslbf**
if (hfCutFilterPresent == 1) {		
hfCornerFreqIndex;	**3**	**uimsbf**
hfFilterStrengthIndex;	**2**	**uimsbf**
}		
hfBoostFilterPresent;	**1**	**bslbf**
if (hfBoostFilterPresent == 1) {		
hfCornerFreqIndex;	**3**	**uimsbf**
hfFilterStrengthIndex;	**2**	**uimsbf**
}		
}		
}		
gainSequenceCount;	**6**	**uimsbf**
gainSetCount;	**6**	**uimsbf**
gainSequenceIndex=-1;		
for (i=0; i<gainSetCount; i++) {		
gainSetIndex=i;		
gainCodingProfile; /* Note (1) */	**2**	**uimsbf**
gainInterpolationType; /* Note (1) */	**1**	**uimsbf**
fullFrame; /* Note (1) */	**1**	**uimsbf**
timeAlignment; /* Note (1) */	**1**	**uimsbf**
timeDeltaMinPresent; /* Note (1) */	**1**	**bslbf**
if (timeDeltaMinPresent==1) {		
bsTimeDeltaMin; /* Note (1) */	**11**	**uimsbf**
}		
if (gainCodingProfile==3) {		
gainSequenceIndex+=1;		
} else {		
bandCount;	**4**	**uimsbf**
if (bandCount>1) {		
drcBandType;	**1**	**uimsbf**
}		
for (j=0; j<bandCount; j++) {		
indexPresent;	**1**	**bslbf**
if (indexPresent==1) {		
gainSequenceIndex**=bsIndex;** /* Note (2) */	**6**	**uimsbf**
} else {		
gainSequenceIndex+=1;		

Syntax	No. of bits	Mnemonic
}		
drcCharacteristicPresent;	**1**	**bslbf**
if (drcCharacteristicPresent==1) {		
drcCharacteristicFormatIsCICP;	**1**	**bslbf**
if (drcCharacteristicFormatIsCICP==1) {		
drcCharacteristic;	**7**	**uimsbf**
} else {		
drcCharacteristicLeftIndex;	**4**	**uimsbf**
drcCharacteristicRightIndex;	**4**	**uimsbf**
}		
} else {		
drcCharacteristic=0;		
drcCharacteristicLeftIndex=0;		
drcCharacteristicRightIndex=0;		
}		
}		
for (j=1; j<bandCount; j++) {		
if (drcBandType==1) {		
crossoverFreqIndex;	**4**	**uimsbf**
} else {		
startSubBandIndex;	**10**	**uimsbf**
}		
}		
}		
}		
}		
The value of this field shall be identical for all DRC gain sets that share one or more DRC gain sequences.		
The values of the index field shall fulfill the condition *bsIndex* < *gainSequenceCount*.		

Table 69 — Syntax of drcCoefficientsUniDrc() payload for ISO/IEC 14496-12

```
aligned(8) class DRCCoefficientsUniDrc extends FullBox('udc2', version, flags=0) {
                                    // N copies of this box, one of these per DRC_location
    bit(2)              reserved = 0;
    signed int(5)       DRC_location;
    bit(1)              drc_frame_size_present;
    if (drc_frame_size_present == 1) {
        bit(1)                  reserved = 0;
        unsigned int(15)        bs_drc_frame_size;
```

```
}
if (version >= 1} {
    bit(5) reserved = 0;
    bit(1) drc_characteristic_left_present;
    bit(1) drc_characteristic_right_present;
    bit(1) shape_filters_present;
    if (drc_characteristic_left_present == 1) {
        bit(4)              reserved = 0;
        unsigned int(4)     characteristic_left_count;
        for (k=1; k<=characteristic_left_count; k++) {
            bit(7)              reserved = 0;
            unsigned int(1)     characteristic_format;
            if (characteristic_format==0) {
                bit(1)              reserved = 0;
                unsigned int(6)     bs_gain_left;
                unsigned int(4)     bs_io_ratio_left;
                unsigned int(4)     bs_exp_left;
                bit(1)              flip_sign_left;
            } else {
                bit(6)              reserved = 0;
                unsigned int(2)     bs_char_node_count;
                for (n=1; n<=bs_char_node_count+1; n++) {
                    bit(3)              reserved = 0;
                    unsigned int(5)     bs_node_level_delta;
                    unsigned int(8)     bs_node_gain;
                }
            }
        }
    }
    if (drc_characteristic_right_present == 1) {
        bit(4)              reserved = 0;
        unsigned int(4)     characteristic_right_count;
        for (k=1; k<=characteristic_right_count; k++) {
            bit(7)              reserved = 0;
            unsigned int(1)     characteristic_format;
            if (characteristic_format==0) {
                bit(1)              reserved = 0;
                unsigned int(6)     bs_gain_right;
                unsigned int(4)     bs_io_ratio_right;
```

```
                unsigned int(4)    bs_exp_right;
                bit(1)             flip_sign_right;
            } else {
                bit(6)             reserved = 0;
                unsigned int(2)    bs_char_node_count;
                for (n=1; n<=bs_char_node_count+1; n++) {
                    bit(3)             reserved = 0;
                    unsigned int(5)    bs_node_level_delta;
                    unsigned int(8)    bs_node_gain;
                }
            }
        }
    }
    if (shape_filters_present==1) {
        bit(4)      reserved = 0;
        unsigned int(4)      shape_filter_count;
        for (k=1; k<=shape_filter_count; k++) {
            bit(4)      reserved = 0;
            bit(1)      LF_cut_filter_present;
            bit(1)      LF_boost_filter_present;
            bit(1)      HF_cut_filter_present;
            bit(1)      HF_boost_filter_present;
            if (LF_cut_filter_present) {
                bit(3)             reserved = 0;
                unsigned int(3)    LF_corner_freq_index;
                unsigned int(2)    LF_filter_strength_index;
            }
            if (LF_boost_filter_present) {
                bit(3)             reserved = 0;
                unsigned int(3)    LF_corner_freq_index;
                unsigned int(2)    LF_filter_strength_index;
            }
            if (HF_cut_filter_present) {
                bit(3)             reserved = 0;
                unsigned int(3)    HF_corner_freq_index;
                unsigned int(2)    HF_filter_strength_index;
            }
            if (HF_boost_filter_present) {
                bit(3)             reserved = 0;
```

```
                    unsigned int(3)    HF_corner_freq_index;
                    unsigned int(2)    HF_filter_strength_index;
                }
            }
        }
    }
    bit(1)                reserved = 0;
    unsigned int(1)    delayMode;
    if (version >= 1} {
        bit(2)                reserved = 0;
        unsigned int(6)    gain_sequence_count;
    }
    unsigned int(6)    gain_set_count;
    for (i=1; i<=gain_set_count; i++) {
        bit(2)                reserved = 0;
        unsigned int(2)    gain_coding_profile;
        unsigned int(1)    gain_interpolation_type;
        unsigned int(1)    full_frame;
        unsigned int(1)    time_alignment;
        bit(1)                time_delta_min_present;
        if (time_delta_min_present == 1) {
            bit(5)                reserved = 0;
            unsigned int(11)    bs_time_delta_min;
        }
        if (gain_coding_profile!=3) {
            bit(3)                reserved = 0;
            unsigned int(4)    band_count;          // shall be >= 1
            unsigned int(1)    drc_band_type;
            for (j = 1; j <= band_count; j++) {
                if (version>=1) {
                    unsigned int(6)    bs_index;
                    bit(1)                drc_characteristic_present
                    bit(1)                drc_characteristic_format_is_CICP
                    if (drc_characteristic_present==1) {
                        if (drc_characteristic_format_is_CICP==1) {
                            bit(1)                reserved;
                            unsigned int(7)    drc_characteristic;
                        } else {
                            unsigned int(4)    drc_characteristic_left_index;
```

```
                                unsigned int(4)    drc_characteristic_right_index;
                            }
                        }
                    } else {
                        bit(1)              reserved = 0;
                        unsigned int(7)     drc_characteristic;
                    }
                }
                for (j = 2; j <= band_count; j++){
                    if (drc_band_type == 1) {
                        bit(4)              reserved = 0;
                        unsigned int(4)     crossover_freq_index;
                    } else {
                        bit(6)                  reserved = 0;
                        unsigned int(10)    start_sub_band_index;
                    }
                }
            }
        }
}
```

Table 70 — Syntax of in-stream drcInstructionsBasic() payload

Syntax	No. of bits	Mnemonic
drcInstructionsBasic()		
{		
drcSetId;	**6**	**uimsbf**
drcLocation;	**4**	**uimsbf**
downmixId;	**7**	**uimsbf**
additionalDownmixIdPresent;	**1**	**bslbf**
if (additionalDownmixIdPresent==1) {		
additionalDownmixIdCount;	**3**	**uimsbf**
for (j=0; j<additionalDownmixIdCount; j++) {		
additionalDownmixId;	**7**	**uimsbf**
}		
} else {		
additionalDownmixIdCount = 0;		
}		
drcSetEffect;	**16**	**bslbf**
if ((drcSetEffect & (3 << 10)) == 0) {		

Syntax	No. of bits	Mnemonic
limiterPeakTargetPresent;	**1**	**bslbf**
if (limiterPeakTargetPresent==1) {		
bsLimiterPeakTarget;	**8**	**uimsbf**
}		
}		
drcSetTargetLoudnessPresent;	**1**	**bslbf**
if (drcSetTargetLoudnessPresent==1) {		
bsDrcSetTargetLoudnessValueUpper;	**6**	**uimsbf**
drcSetTargetLoudnessValueLowerPresent;	**1**	**bslbf**
if (drcSetTargetLoudnessValueLowerPresent==1) {		
bsDrcSetTargetLoudnessValueLower;	**6**	**uimsbf**
}		
}		
}		

Table 71 — Syntax of drcInstructionsBasic() payload for ISO/IEC 14496-12

```
aligned(8) class DRCInstructionsBasic extends FullBox('udi1', version, flags=0) {
                    // N copies, one for each overall combination DRC that can be applied
    bit(3)              reserved = 0;
    unsigned int(6)     DRC_set_ID;             // shall be non-zero and unique
    signed int(5)       DRC_location;
    unsigned int(7)     downmix_ID;             // if 0 – to base
        // if non-0, applies after downmix
        // if 0x7f, applies before or after downmix unsigned int(3) additional_downmix_ID_count;
    for (j=0; j<additional_downmix_ID_count; j++) {
        bit(1)          reserved = 0;
        unsigned int(7)     additional_downmix_ID;
    }
    bit(16) DRC_set_effect;
    if ((DRC_set_effect & (3<<10)) == 0) {
        bit(7)    reserved = 0;
        bit(1)    limiter_peak_target_present;
        if (limiter_peak_target_present == 1) {
            unsigned int(8)    bs_limiter_peak_target;
        }
    }
    bit(7)    reserved = 0;
    bit(1)    DRC_set_target_loudness_present;
```

```
    if (DRC_set_target_loudness_present==1) {
        bit(4)              reserved = 0;
        unsigned int(6)     bs_DRC_set_target_loudness_value_upper;
        unsigned int(6)     bs_DRC_set_target_loudness_value_lower;
    }
}
```

Table 72 — Syntax of in-stream drcInstructionsUniDrc() payload

Syntax	No. of bits	Mnemonic
drcInstructionsUniDrc()		
{		
drcSetId;	**6**	**uimsbf**
drcLocation;	**4**	**uimsbf**
downmixId;	**7**	**uimsbf**
additionalDownmixIdPresent;	**1**	**bslbf**
if (additionalDownmixIdPresent==1) {		
additionalDownmixIdCount;	**3**	**uimsbf**
for (j=0; j<additionalDownmixIdCount; j++) {		
additionalDownmixId;	**7**	**uimsbf**
}		
} else {		
additionalDownmixIdCount = 0;		
}		
drcSetEffect;	**16**	**bslbf**
if ((drcSetEffect & (3<<10)) == 0) {		
limiterPeakTargetPresent;	**1**	**bslbf**
if (limiterPeakTargetPresent==1) {		
bsLimiterPeakTarget;	**8**	**uimsbf**
}		
}		
drcSetTargetLoudnessPresent;	**1**	**bslbf**
if (drcSetTargetLoudnessPresent==1) {		
bsDrcSetTargetLoudnessValueUpper;	**6**	**uimsbf**
drcSetTargetLoudnessValueLowerPresent;	**1**	**bslbf**
if (drcSetTargetLoudnessValueLowerPresent==1) {		
bsDrcSetTargetLoudnessValueLower;	**6**	**uimsbf**
}		
}		
dependsOnDrcSetPresent;	**1**	**bslbf**

Syntax	No. of bits	Mnemonic
if (dependsOnDrcSetPresent==1) {		
dependsOnDrcSet;	**6**	**uimsbf**
} else {		
noIndependentUse;	**1**	**bslbf**
}		
channelCount = baseChannelCount;		
if ((drcSetEffect & (3<<10)) != 0) {		
for (i=0; i<channelCount; i++) {		
bsGainSetIndex;	**6**	**uimsbf**
duckingScalingPresent;	**1**	**bslbf**
if (duckingScalingPresent==1) {		
bsDuckingScaling;	**4**	**bslbf**
}		
repeatParameters;	**1**	**bslbf**
if (repeatParameters) {		
repeatParametersCount **=bsRepeatParametersCount**+1;	**5**	**uimsbf**
i = i + repeatParametersCount;		
}		
}		
} else {		
if (downmixId!=0 && downmixId!=0x7F && additionalDownmixIdCount==0) {		
channelCount = targetChannelCountFromDownmixId;		
} else if (downmixId==0x7F || additionalDownmixIdCount!=0) {		
channelCount = 1;		
}		
for (i=0; i<channelCount; i++) {		
bsGainSetIndex;	**6**	**uimsbf**
repeatGainSetIndex;	**1**	**bslbf**
if (repeatGainSetIndex) {		
repeatGainSetIndexCount=**bsRepeatGainSetIndexCount**+1;	**5**	**uimsbf**
i = i + repeatGainSetIndexCount;		
}		
}		
for (i=0; i<nDrcChannelGroups; i++) { /* NOTE 1 */		
gainScalingPresent;	**1**	**bslbf**
if (gainScalingPresent==1) {		
bsAttenuationScaling;	**4**	**uimsbf**
bsAmplificationScaling;	**4**	**uimsbf**

Syntax	No. of bits	Mnemonic
}		
gainOffsetPresent;	**1**	**bslbf**
if (gainOffsetPresent==1) {		
bsGainOffset;	**6**	**bslbf**
}		
}		
}		
NOTE The number of channel groups *nDrcChannelGroups* is calculated according to Table 15.		

Table 73 — Syntax of in-stream drcInstructionsUniDrcV1() payload

Syntax	No. of bits	Mnemonic
drcInstructionsUniDrcV1()		
{		
drcSetId;	**6**	**uimsbf**
drcSetComplexityLevel;	**4**	**uimsbf**
drcLocation;	**4**	**uimsbf**
channelCount = baseChannelCount;		
downmixIdPresent;	**1**	**bslbf**
if (downmixIdPresent==1) {		
downmixId;	**7**	**uimsbf**
drcApplyToDownmix;	**1**	**bslbf**
additionalDownmixIdPresent;	**1**	**bslbf**
if (additionalDownmixIdPresent==1) {		
additionalDownmixIdCount;	**3**	**uimsbf**
for (j=0; j<additionalDownmixIdCount; j++) {		
additionalDownmixId;	**7**	**uimsbf**
}		
} else {		
additionalDownmixIdCount = 0;		
}		
if ((drcApplyToDownmix == 1) && (downmixId != 0) &&		
(downmixId != 0x7F) && (additionalDownmixIdCount==0)) {		
channelCount = targetChannelCountFromDownmix;		
} else if ((drcApplyToDownmix == 1) && ((downmixId == 0x7F) ||		
(additionalDownmixIdCount!=0))) {		
channelCount = 1;		
}		

Syntax	No. of bits	Mnemonic
} else {		
downmixId = 0;		
}		
drcSetEffect;	**16**	**bslbf**
if ((drcSetEffect & (3<<10)) == 0) {		
limiterPeakTargetPresent;	**1**	**bslbf**
if (limiterPeakTargetPresent==1) {		
bsLimiterPeakTarget;	**8**	**uimsbf**
}		
}		
drcSetTargetLoudnessPresent;	**1**	**bslbf**
if (drcSetTargetLoudnessPresent==1) {		
bsDrcSetTargetLoudnessValueUpper;	**6**	**uimsbf**
drcSetTargetLoudnessValueLowerPresent;	**1**	**bslbf**
if (drcSetTargetLoudnessValueLowerPresent==1) {		
bsDrcSetTargetLoudnessValueLower;	**6**	**uimsbf**
}		
}		
dependsOnDrcSetPresent;	**1**	**bslbf**
if (dependsOnDrcSetPresent==1) {		
dependsOnDrcSet;	**6**	**uimsbf**
} else {		
noIndependentUse;	**1**	**bslbf**
}		
requiresEq;	**1**	**bslbf**
if ((drcSetEffect & (3<<10)) != 0) {		
for (i=0; i<baseChannelCount; i++) {		
bsGainSetIndex;	**6**	**uimsbf**
duckingScalingPresent;	**1**	**bslbf**
if (duckingScalingPresent==1) {		
bsDuckingScaling;	**4**	**bslbf**
}		
repeatParameters;	**1**	**bslbf**
if (repeatParameters==1) {		
bsRepeatParametersCount;	**5**	**uimsbf**
repeatGainSetIndexCount=bsRepeatGainSetIndexCount+1;		
i = i + repeatParametersCount;		
}		

Syntax	No. of bits	Mnemonic
}		
} else {		
for (i=0; i<channelCount; i++) {		
bsGainSetIndex;	**6**	**uimsbf**
repeatGainSetIndex;	**1**	**bslbf**
if (repeatGainSetIndex==1) {		
bsRepeatGainSetIndexCount;	**5**	**uimsbf**
repeatGainSetIndexCount=bsRepeatGainSetIndexCount+1;		
i = i + repeatGainSetIndexCount;		
}		
}		
for (i=0; i<nDrcChannelGroups; i++) { /* NOTE 1 */		
for (k=0; k<bandCount; k++) {		
targetCharacteristicLeftPresent;	**1**	**bslbf**
if (targetCharacteristicLeftPresent==1) {		
targetCharacteristicLeftIndex;	**4**	**uimsbf**
}		
targetCharacteristicRightPresent;	**1**	**bslbf**
if (targetCharacteristicRightPresent==1) {		
targetCharacteristicRightIndex;	**4**	**uimsbf**
}		
gainScalingPresent;	**1**	**bslbf**
if (gainScalingPresent==1) {		
bsAttenuationScaling;	**4**	**uimsbf**
bsAmplificationScaling;	**4**	**uimsbf**
}		
gainOffsetPresent;	**1**	**bslbf**
if (gainOffsetPresent==1) {		
bsGainOffset;	**6**	**bslbf**
}		
}		
if (bandCount==1) {		
shapeFilterPresent;	**1**	**bslbf**
if (shapeFilterPresent==1) {		
shapeFilterIndex;	**4**	**bslbf**
}		
}		
}		

Syntax	No. of bits	Mnemonic
}		
}		
NOTE The number of channel groups *nDrcChannelGroups* is calculated according to Table 15.		

Table 74 — Syntax of drcInstructionsUniDrc() payload for ISO/IEC 14496-12

```
aligned(8) class DRCInstructionsUniDrc extends FullBox('udi2', version, flags=0) {
        // if version == 0, N copies, one for each overall combination DRC
    if (version>=1) {
        bit(2)              reserved = 0;
        unsigned int(6)     DRC_instructions_count;
    else {
        unsigned int        DRC_instructions_count = 1;
    }
    for (a=1; a<=DRC_instructions_count; a++) {
        if (version==0) {
            bit(3)              reserved = 0;
            unsigned int(6)     DRC_set_ID;         // shall be non-zero and unique
            signed int(5)       DRC_location;
            unsigned int(7)     downmix_ID;         // if 0 – to base
                                                    // if non-0, applies after downmix
                                                    // if 0x7f, applies before or after downmix
            unsigned int(3)     additional_downmix_ID_count;
            for (j=1; j<=additional_downmix_ID_count; j++) {
                bit(1)              reserved = 0;
                unsigned int(7)     additional_downmix_ID;
            }
        } else {
            unsigned int(6)     DRC_set_ID;         // shall be non-zero and unique
            unsigned int(4)     DRC_complexity_level;
            signed int(5)       DRC_location;
            unsigned int(1)     downmix_ID_present;
            if (downmix_ID_present==1) {
                bit(5)              reserved = 0;
                unsigned int(7)     downmix_ID;
                unsigned int(1)     DRC_apply_to_downmix
                unsigned int(3)     additional_downmix_ID_count;
                for (j=1; j<=additional_downmix_ID_count; j++) {
                    bit(1)              reserved = 0;
```

```
                    unsigned int(7)    additional_downmix_ID;
                }
            }
        }
        unsigned int(16)    DRC_set_effect;
        if ((DRC_set_effect & (3<<10)) == 0) {
            bit(7)    reserved = 0;
            bit(1)    limiter_peak_target_present;
            if (limiter_peak_target_present == 1) {
                unsigned int(8)    bs_limiter_peak_target;
            }
        }
        bit(7)    reserved = 0;
        bit(1)    DRC_set_target_loudness_present;
        if (DRC_set_target_loudness_present==1) {
            bit(4)                reserved = 0;
            unsigned int(6)    bs_DRC_set_target_loudness_value_upper;
            unsigned int(6)    bs_DRC_set_target_loudness_value_lower;
        }
        if (version>=1) {
            bit(1)    requiresEq;
        } else {
            bit(1)    reserved = 0;
        }
        unsigned int(6)    depends_on_DRC_set;
                // if non-zero, shall match a DRC that shall be applied before this one
                // it shall be examined to see if that is before/after downmix
        if (depends_on_DRC_set==0) {
            bit(1)    no_independent_use;
        } else {
            bit(1)    reserved = 0;
        }
        unsigned int(8)    channel_count;
            //if version==0:
            // - shall match channel count of downmix if
            //       downmix_ID!=0x0 AND downmix_ID!=0x7F AND
            //       additional_downmix_ID_count!=0
            // - shall be 1 and applied to all channels if
            //       downmix_ID==0x7F OR additional_downmix_ID_count!=0,
```

```
        // - else shall match channel count of the base layout
        //if version>=1:
        // - shall match channel count of downmix if
        //      drcApplyToDownmix==1 AND downmix_ID!=0x0 AND
        //      downmix_ID!=0x7F AND additional_downmix_ID_count==0,
        // - shall be 1 and applied to all channels if
        //      drcApplyToDownmix==1 AND (downmix_ID==0x7F OR
        //      additional_downmix_ID_count!=0),
        // - else shall match channel count of the base layout
    for (i=1; i<=channel_count; i++){
        unsigned int (8) channel_group_index;
                // 1-based channel_group_index for channel
    }
    for (i=1; i<=channel_group_count; i++){
        bit(2)              reserved = 0;
        unsigned int(6)   bs_gain_set_index;
                    // if 0, then this channel_group/object is not processed
                    // else 1-based index into the DRC gain set array for this location
    }
    if ((DRC_set_effect & (3<<10)) != 0) {
        for (i=1; i<=channel_group_count; i++) {
            bit(7)   reserved = 0;
            bit(1)   ducking_scaling_present;
            if (ducking_scaling_present==1) {
                bit(4)   reserved = 0;
                bit(4)   bs_ducking_scaling;
            }
        }
    } else {
        for (i=1; i<=channel_group_count; i++) {
            if (version >= 1) {
                bit(4)              reserved = 0;
                unsigned int(4)   band_count;
                for (k=1; k<=band_count; k++) {
                    bit(4)   reserved = 0;
                    bit(1)   target_characteristic_left_present;
                    bit(1)   target_characteristic_right_present;
                    bit(1)   gain_scaling_present;
                    bit(1)   gain_offset_present;
```

```
            if (target_characteristic_left_present ==1) {
                bit(4)              reserved = 0;
                unsigned int(4)     target_characteristic_left_index;
            }
            if (target_characteristic_right_present ==1) {
                bit(4)              reserved = 0;
                unsigned int(4)     target_characteristic_right_index;
            }
            if (gain_scaling_present==1) {
                unsigned int(4)     bs_attenuation_scaling;
                unsigned int(4)     bs_amplification_scaling;
            }
            if (gain_offset_present==1) {
                bit(2)              reserved = 0;
                unsigned int(6)     bs_gain_offset;
            }
        }
        if (band_count==1) {
            bit(1)      shape_filter_present;
            if (shape_filter_present==1) {
                bit(3)              reserved = 0;
                unsigned int(4)     shape_filter_index;
            } else {
                bit(7)      reserved = 0;
            }
        }
    } else {
        bit(7)      reserved = 0;
        bit(1)      gain_scaling_present;
        if (gain_scaling_present==1) {
            unsigned int(4)     bs_attenuation_scaling;
            unsigned int(4)     bs_amplification_scaling;
        }
        bit(7)      reserved = 0;
        bit(1)      gain_offset_present;
        if (gain_offset_present==1) {
            bit(2)      reserved = 0;
            bit(6)      bs_gain_offset;
        }
```

```
                    }
                }
            }
        }
}
```

Table 75 — Syntax of uniDrcConfigExtension() payload

Syntax	No. of bits	Mnemonic
uniDrcConfigExtension()		
{		
while (**uniDrcConfigExtType !**= UNIDRCCONFEXT_TERM) {	**4**	**uimsbf**
extSizeBits = **bitSizeLen** + 4;	**4**	**uimsbf**
extBitSize = **bitSize** + 1;	**extSizeBits**	**uimsbf**
switch (uniDrcConfigExtType) {		
case UNIDRCCONFEXT_PARAM_DRC:		
drcCoefficientsParametricDrc();		
parametricDrcInstructionsCount;	**4**	**uimsbf**
for (i=0; i<parametricDrcInstructionsCount; i++) {		
parametricDrcInstructions ();		
}		
break;		
case UNIDRCCONFEXT_V1:		
downmixInstructionsV1Present;	**1**	**bslbf**
if (downmixInstructionsV1Present==1) {		
downmixInstructionsV1Count;	**7**	**uimsbf**
for (i=0; i<downmixInstructionsV1Count; i++) {		
downmixInstructionsV1();		
}		
}		
drcCoeffsAndInstructionsUniDrcV1Present;	**1**	**bslbf**
if (drcCoeffsAndInstructionsUniDrcV1Present==1) {		
drcCoefficientsUniDrcV1Count;	**3**	**uimsbf**
for (i=0; i<drcCoefficientsUniDrcV1Count; i++) {		
drcCoefficientsUniDrcV1();		
}		
drcInstructionsUniDrcV1Count;	**6**	**uimsbf**
for (i=0; i<drcInstructionsUniDrcV1Count; i++) {		
drcInstructionsUniDrcV1();		
}		

Syntax	No. of bits	Mnemonic
}		
loudEqInstructionsPresent;	**1**	**bslbf**
if (loudEqInstructionsPresent==1) {		
loudEqInstructionsCount;	**4**	**uimsbf**
for (i=0; i<loudEqInstructionsCount; i++) {		
loudEqInstructions();		
}		
}		
eqPresent;	**1**	**bslbf**
if (eqPresent==1) {		
eqCoefficients();		
eqInstructionsCount;	**4**	**uimsbf**
for (i=0; i<eqInstructionsCount; i++) {		
eqInstructions();		
}		
}		
break;		
/* add future extensions here */		
default:		
for (i=0; i<extBitSize; i++) {		
otherBit;	**1**	**bslbf**
}		
}		
}		
}		

Table 76 — Syntax of uniDrcConfigExtension() payload for ISO/IEC 14496-12

```
aligned(8) class UniDrcConfigExtension extends Box('udex') {
        // the ordering of the following boxes shall be maintained
        // we permit one loudness EQ instruction box:
    LoudEQInstructions();
        // we permit one EQ coefficients box:
    EQCoefficients();
        // we permit one EQ instruction box:
    EQInstructions();
        // we permit one parametric DRC coefficients box:
    DRCCoefficientsParametricDrc();
        // we permit one parametric DRC instruction box:
```

```
    ParametricDrcInstructions ();
    Box ();          // further boxes as needed
}
```

7.4 Syntax of DRC gain sequence

The syntax of drcGainSequence() payload is shown in Table 77.

Table 77 — Syntax of drcGainSequence() payload

Syntax	No. of bits	Mnemonic
drcGainSequence()		
{		
drcGainCodingMode;	**1**	**uimsbf**
if (drcGainCodingMode == 0) {		
gainInitialCode;	**1..11**	**vlclbf**
nNodes = 1;		
}		
else {		
k=0;		
do {		
k++;		
}		
while (**endMarker**[k-1] != 1);	**1**	**uimsbf**
nNodes = k;		
if (gainInterpolationType == 0) {		
for (k=0; k<nNodes; k++) {		
slopeCode[k];	**1..9**	**vlclbf**
}		
}		
if (fullFrame == 0) {		
frameEndFlag;	**1**	**uimsbf**
}		
else {		
frameEndFlag = 1;		
}		
for (k=0; k<nNodes-1; k++) {		
timeDeltaCode[k];	**2..Z+2**	**vlclbf**
}		
if (frameEndFlag == 0) {		
k=nNodes-1;		

Syntax	No. of bits	Mnemonic
timeDeltaCode[k];	**2..Z+2**	**vlclbf**
}		
gainInitialCode;	**1..11**	**vlclbf**
for (k=1; k<nNodes; k++) {		
gainDeltaCode[k-1];	**1..11**	**vlclbf**
}		
}		
}		

7.5 Syntax of parametric DRC tool

Tables 78 - 83 show the syntax of the parametric DRC tool.

Table 78 — Syntax of drcCoefficientsParametricDrc() payload

Syntax	No. of bits	Mnemonic
drcCoefficientsParametricDrc()		
{		
drcLocation;	**4**	**uimsbf**
parametricDrcFrameSizeFormat;	**1**	**bslbf**
if (parametricDrcFrameSizeFormat==0) {		
bsParametricDrcFrameSize;	**4**	**uimsbf**
} else {		
bsDrcFrameSize;	**15**	**uimsbf**
}		
parametricDrcDelayMaxPresent;	**1**	**bslbf**
if (parametricDrcDelayMaxPresent==1) {		
bsParametricDrcDelayMax;	**8**	**uimsbf**
}		
resetParametricDrc;	**1**	**bslbf**
parametricDrcGainSetCount;	**6**	**uimsbf**
for (i=0; i<parametricDrcGainSetCount; i++) {		
parametricDrcId;	**4**	**uimsbf**
sideChainConfigType;	**3**	**bslbf**
if (sideChainConfigType==1) {		
downmixId;	**7**	**uimsbf**
levelEstimChannelWeightFormat;	**1**	**bslbf**
for (j=0; j<channelCountFromDownmixId; j++) {		
if (levelEstimChannelWeightFormat==0) {		
addChannel[j];	**1**	**bslbf**

Syntax	No. of bits	Mnemonic
} else {		
bsChannelWeight[j];	**4**	**uimsbf**
}		
}		
}		
drcInputLoudnessPresent;	**1**	**bslbf**
if (drcInputLoudnessPresent==1) {		
bsDrcInputLoudness;	**8**	**uimsbf**
} else {		
/* provided by loudnessInfoSet() */		
}		
}		
}		

Table 79 — Syntax of drcCoefficientsParametricDrc() payload for ISO/IEC 14496-12

```
aligned(8) class DRCCoefficientsParametricDrc extends FullBox('pdc1', version=0, flags=0) {
    bit(1)              reserved = 0;
    signed int(5)       drc_location;
    bit(1)              parametric_drc_frame_size_format;
    bit(1)              reset_parametric_drc;
    if (parametric_drc_frame_size_format==0) {
        bit(4)                  reserved = 0;
        unsigned int(4)     bs_parametric_drc_frame_size;
    } else {
        bit(1)                  reserved = 0;
        unsigned int(15)    bs_drc_frame_size;
    }
    bit(1)    reserved = 0;
    bit(1)    parametric_drc_delay_max_present;
    if (parametric_drc_delay_max_present==1) {
        unsigned int (8)    parametric_drc_delay_max;
    }
    unsigned int(6)    gain_set_count;                  // parametric_drc_gain_set_count
    for (gain_set_index=1; gain_set_index <= gain_set_count; gain_set_index++) {
        unsigned int(4)    parametric_drc_ID;
        bit(1)                  drc_input_loudness_present;
        bit(3)                  side_chain_config_type;
        if (drc_input_loudness_present==1) {
```

```
            unsigned int(8)     bs_drc_input_loudness;
        } else {
            // provided by LoudnessBox
        }
        if (side_chain_config_type==1) {
            bit(1)              reserved = 0;
            unsigned int(7)     downmix_ID;
            bit(1)              reserved = 0;
            unsigned int(7)     channel_count_from_downmix_ID;      // shall match
            for (j=1; j <= channel_count_from_downmix_ID; j++) {
                unsigned int(4)     bs_channel_weight;
            }
        }
    }
}
```

Table 80 — Syntax of parametricDrcInstructions() payload

Syntax	No. of bits	Mnemonic
parametricDrcInstructions ()		
{		
parametricDrcId;	**4**	**uimsbf**
parametricDrcLookAheadPresent;	**1**	**bslbf**
if (parametricDrcLookAheadPresent==1) {		
bsParametricDrcLookAhead;	**7**	**uimsbf**
}		
parametricDrcPresetIdPresent;	**1**	**bslbf**
if (parametricDrcPresetIdPresent==1) {		
parametricDrcPresetId;	**7**	**uimsbf**
} else {		
parametricDrcType;	**3**	**uimsbf**
if (parametricDrcType == PARAM_DRC_TYPE_FF) {		
parametricDrcTypeFeedForward();		
} else if (parametricDrcType == PARAM_DRC_TYPE_LIM) {		
parametricDrcTypeLimiter();		
} else {		
lenSizeBits = **bitSizeLen** + 4;	**3**	**uimsbf**
lenBitSize = **bitSize** + 1;	**lenSizeBits**	**uimsbf**
switch (parametricDrcType) {		
/* add future extensions here */		

Syntax	No. of bits	Mnemonic
default:		
for (i=0; i<lenBitSize; i++) {		
otherBit;	**1**	**bslbf**
}		
break;		
}		
}		
}		
}		

Table 81 — Syntax of parametricDrcInstructions() payload for ISO/IEC 14496-12

```
aligned(8) class ParametricDrcInstructions extends FullBox('pdi1', version=0, flags=0) {
    bit(4)              reserved = 0;
    unsigned int(4)     parametric_DRC_instructions_count;
    for (a=1; a<=parametric_DRC_instructions_count; a++) {
        bit(2)              reserved = 0;
        unsigned in(4)      parametric_drc_ID;
        bit(1)              parametric_drc_look_ahead_present;
        bit(1)              parametric_drc_preset_ID_present;
        if (parametric_drc_look_ahead_present==1) {
            bit(1)              reserved = 0;
            unsigned int(7)     bs_parametric_drc_look_ahead;
        }
        if (parametric_drc_preset_ID_present==1) {
            bit(1)              reserved = 0;
            unsigned int(7)     parametric_drc_preset_ID;
        } else {
            bit(5)              reserved = 0;
            unsigned int(3)     parametric_drc_type;
            if (parametric_drc_type == 0) { /* PARAM_DRC_TYPE_FF */
                bit(3)              reserved = 0;
                unsigned int(2)     level_estim_K_weighting_type;
                bit(1)              level_estim_integration_time_present;
                unsigned int(1)     drc_curve_definition_type;
                bit(1)              drc_gain_smooth_parameters_present;
                if (level_estim_integration_time_present == 1) {
                    bit(2)              reserved = 0;
                    unsigned int(6)     bs_level_estim_integration_time;
```

```
    }
    if (drc_curve_definition_type==0) {
        bit(1)              reserved = 0;
        unsigned int(7)     DRC_characteristic;
    } else {
        bit(5)              reserved = 0;
        unsigned int(3)     bs_node_count;      // node_count=bs_node_count+2
        for (j = 1; j <= node_count; j++) {
            if (j==1) {
                bit(2)              reserved = 0;
                unsigned int(6)     bs_node_level_initial;
            } else {
                bit(3)              reserved = 0;
                unsigned int(5)     bs_node_level_delta;
            }
            bit(2)              reserved = 0;
            unsigned int(6)     bs_node_gain;
        }
    }
    if (drc_gain_smooth_parameters_present==0) {
        // default or matching to drcCharacteristic if available
    } else {
        bit(6)              reserved = 0;
        bit(1)              gain_smooth_time_fast_present;
        bit(1)              gain_smooth_hold_off_count_present;
        unsigned int(8)     bs_gain_smooth_attack_time_slow;
        unsigned int(8)     bs_gain_smooth_release_time_slow;
        if (gain_smooth_time_fast_present==1) {
            unsigned int(8)     bs_gain_smooth_attack_time_fast;
            unsigned int(8)     bs_gain_smooth_release_time_fast;
            bit(7)     reserved = 0;
            bit(1)     gain_smooth_threshold_present;
            if (gain_smooth_threshold_present==1){
                bit(3)              reserved = 0;
                unsigned int(5)     bs_gain_smooth_attack_threshold;
                bit(3)              reserved = 0;
                unsigned int(5)     bs_gain_smooth_release_threshold;
            }
        }
```

```
                    if (gain_smooth_hold_off_count_present==1) {
                        bit(1)              reserved = 0;
                        unsigned int(7)     bs_gain_smooth_hold_off;
                    }
                }
            } else if (parametric_drc_type == 1) {          /* PARAM_DRC_TYPE_LIM */
                bit(6)     reserved = 0;
                bit(1)     parametric_lim_threshold_present;
                bit(1)     parametric_lim_release_time_present;
                if (parametric_lim_threshold_present==1) {
                    unsigned int(8)     bs_parametric_lim_threshold;
                }
                parametric_lim_attack_time = parametric_drc_look_ahead;
                if (parametric_lim_release_time_present==1) {
                    unsigned int(8)     bs_parametric_lim_release_time;
                }
            } else {
                // disable parametric DRC
            }
        }
    }
}
```

Table 82 — Syntax of parametricDrcTypeFeedForward() payload

Syntax	No. of bits	Mnemonic
parametricDrcTypeFeedForward()		
{		
levelEstimKWeightingType;	**2**	**uimsbf**
levelEstimIntegrationTimePresent;	**1**	**bslbf**
if (levelEstimationIntegrationTimePresent==1) {		
bsLevelEstimIntegrationTime;	**6**	**uimsbf**
}		
drcCurveDefinitionType;	**1**	**bslbf**
if (drcCurveDefintionType==0) {		
drcCharacteristic; /* according to CICP */	**7**	**uimsbf**
} else {		
bsNodeCount;	**3**	**uimsbf**
for(j=0; j<nodeCount; j++) {		
if (j==0) {		

Syntax	No. of bits	Mnemonic
bsNodeLevelInitial;	**6**	**uimsbf**
} else {		
bsNodeLevelDelta;	**5**	**uimsbf**
}		
bsNodeGain;	**6**	**uimsbf**
}		
}		
drcGainSmoothParametersPresent;	**1**	**bslbf**
if (drcGainSmoothParametersPresent==0) {		
/* default or matching to drcCharacteristic if available */		
} else {		
bsGainSmoothAttackTimeSlow;	**8**	**uimsbf**
bsGainSmoothReleaseTimeSlow;	**8**	**uimsbf**
gainSmoothTimeFastPresent;	**1**	**bslbf**
if (gainSmoothTimeFastPresent==1) {		
bsGainSmoothAttackTimeFast;	**8**	**uimsbf**
bsGainSmoothReleaseTimeFast;	**8**	**uimsbf**
gainSmoothTresholdPresent;	**1**	**bslbf**
if (gainSmoothTresholdPresent==1) {		
bsGainSmoothAttackThreshold;	**5**	**uimsbf**
bsGainSmoothReleaseThreshold;	**5**	**uimsbf**
}		
}		
gainSmoothHoldOffCountPresent;	**1**	**bslbf**
if (gainSmoothHoldOffCountPresent==1) {		
bsGainSmoothHoldOff;	**7**	**uimsbf**
}		
}		
}		

Table 83 — Syntax of parametricDrcTypeLimiter() payload

Syntax	No. of bits	Mnemonic
parametricDrcTypeLimiter()		
{		
parametricLimThresholdPresent;	**1**	**bslbf**
if (parametricLimThresholdPresent==1) {		
bsParametricLimThreshold;	**8**	**uimsbf**
}		

Syntax	No. of bits	Mnemonic
parametricLimAttackTime = parametricDrcLookAhead;		
parametricLimReleaseTimePresent;	**1**	**bslbf**
if (parametricLimReleaseTimePresent==1) {		
bsParametricLimReleaseTime;	**8**	**uimsbf**
}		
}		

7.6 Syntax of equalization tools

Tables 84 - 90 show the syntax of equalization tools.

Table 84 — Syntax of in-stream loudEqInstructions() payload

Syntax	No. of bits	Mnemonic
loudEqInstructions()		
{		
loudEqSetId;	**4**	**uimsbf**
drcLocation;	**4**	**uimsbf**
downmixIdPresent;	**1**	**bslbf**
if (downmixIdPresent==1) {		
downmixId;	**7**	**uimsbf**
additionalDownmixIdPresent;	**1**	**bslbf**
if (additionalDdownmixIdPresent==1) {		
additionalDownmixIdCount;	**7**	**uimsbf**
for (i=0; i<additionalDownmixIdCount; i++) {		
additionalDownmixId;	**7**	**uimsbf**
}		
} else {		
additionalDownmixIdCount = 0;		
}		
} else {		
downmixId = 0;		
}		
drcSetIdPresent;	**1**	**bslbf**
if (drcSetIdPresent==1) {		
drcSetId;	**6**	**uimsbf**
additionalDrcSetIdPresent;	**1**	**bslbf**
if (additionalDrcSetIdPresent ==1) {		
additionalDrcSetIdCount;	**6**	**uimsbf**
for (i=0; i<additionalDrcSetIdCount; i++) {		
additionalDrcSetId;	**6**	**uimsbf**
}		

Syntax	No. of bits	Mnemonic
} else {		
additionalDrcSetIdCount = 0;		
}		
} else {		
drcSetId = 0;		
}		
eqSetIdPresent;	**1**	**bslbf**
if (eqSetIdPresent==1) {		
eqSetId;	**6**	**uimsbf**
additionalEqSetIdPresent;	**1**	**bslbf**
if (additionalEqSetIdPresent ==1) {		
additionalEqSetIdCount;	**6**	**uimsbf**
for (i=0; i<additionalEqSetIdCount; i++) {		
additionalEqSetId;	**6**	**uimsbf**
}		
} else {		
additionalEqSetIdCount = 0;		
}		
} else {		
eqSetId = 0;		
}		
loudnessAfterDrc;	**1**	**bslbf**
loudnessAfterEq;	**1**	**bslbf**
loudEqGainSequenceCount;	**6**	**uimsbf**
for (i=0; i<loudEqGainSequenceCount; i++) {		
gainSequenceIndex;	**6**	**uimsbf**
drcCharacteristicFormatIsCICP;	**1**	**bslbf**
if (drcCharacteristicFormatIsCICP==1) {		
drcCharacteristic;	**7**	**uimsbf**
} else {		
drcCharacteristicLeftIndex;	**4**	**uimsbf**
drcCharacteristicRightIndex;	**4**	**uimsbf**
}		
frequencyRangeIndex;	**6**	**uimsbf**
bsLoudEqScaling;	**3**	**uimsbf**
bsLoudEqOffset;	**5**	**uimsbf**
}		
}		

Table 85 — Syntax of loudEqInstructions() payload for ISO/IEC 14496-12

```
aligned(8) class LoudEqInstructions extends FullBox('leqi', version=0, flags=0) {
    bit(4)              reserved = 0;
    unsigned int (4)    loud_EQ_instructions_count;
    for (a=1; a<=loud_EQ_instructions_count; a++) {
        bit(2)              reserved = 0;
        unsigned int(4)     loud_EQ_set_ID;
        signed int(5)       drc_location;
        bit(1)              downmix_ID_present;
        bit(1)              DRC_set_ID_present;
        bit(1)              EQ_set_ID_present;
        bit(1)              loudness_after_EQ;
        bit(1)              loudness_after_DRC;
        if (downmix_ID_present==1) {
            unsigned int(7)     downmix_ID;
            bit(1)              additional_downmix_ID_present;
            if (additional_downmix_ID_present==1) {
                bit(1)              reserved = 0;
                unsigned int(7)     additional_downmix_ID_count;
                for (i=1; i<=additional_downmix_ID_count; i++) {
                    bit(1)              reserved = 0;
                    unsigned int(7)     additional_downmix_ID;
                }
            }
        }
        if (DRC_set_ID_present ==1) {
            bit(1)              reserved = 0;
            unsigned int(6)     DRC_set_ID;
            bit(1)              additional_DRC_set_ID_present;
            if (additional_DRC_set_ID_present==1) {
                bit(2)              reserved = 0;
                unsigned int(6)     additional_DRC_set_ID_count;
                for (i=1; i<=additional_DRC_set_ID_count; i++) {
                    bit(2)              reserved = 0;
                    unsigned int(6)     additional_DRC_set_ID;
                }
            }
        }
        if (EQ_set_ID_present ==1) {
```

```
            bit(1)              reserved = 0;
            unsigned int(6)     EQ_set_ID;
            bit(1)              additional_EQ_set_ID_present;
            if (additional_EQ_set_ID_present==1) {
                bit(2)              reserved = 0;
                unsigned int(6)     additional_EQ_set_ID_count;
                for (i=1; i<=additional_EQ_set_ID_count; i++) {
                    bit(2)              reserved = 0;
                    unsigned int(6)     additional_EQ_set_ID;
                }
            }
        }
        bit(2)              reserved = 0;
        unsigned int(6)     loud_EQ_gain_sequence_count;
        for (i=1; i<=loud_EQ_gain_sequence_count; i++) {
            bit(1)              reserved = 0;
            unsigned int(6)     gain_sequence_index;
            bit(1)              DRC_characteristic_format_is_CICP;
            if (DRC_characteristic_format_is_CICP==1) {
                bit(1)              reserved = 0;
                unsigned int(7)     DRC_characteristic;
            } else {
                unsigned int(4)     DRC_characteristic_left_index;
                unsigned int(4)     DRC_characteristic_right_index;
            }
            bit(2)              reserved = 0;
            unsigned int(6)     frequency_range_index;
            unsigned int(3)     bs_loud_eq_scaling;
            unsigned int(5)     bs_loud_eq_offset;
        }
    }
}
```


Table 86 — Syntax of in-stream eqCoefficients() payload

Syntax	No. of bits	Mnemonic
eqCoefficients()		
{		
eqDelayMaxPresent;	**1**	**bslbf**
if (eqDelayMaxPresent==1) {		

Syntax	No. of bits	Mnemonic
bsEqDelayMax;	**8**	**uimsbf**
}		
uniqueFilterBlockCount;	**6**	**uimsbf**
for (j=0; j<uniqueFilterBlockCount; j++) {		
filterBlockIndex = j;		
filterElementCount;	**6**	**uimsbf**
for (k=0; k<filterElementCount; k++) {		
filterElementIndex;	**6**	**uimsbf**
filterElementGainPresent;	**1**	**bslbf**
if (filterElementGainPresent == 1) {		
bsFilterElementGain;	**10**	**uimsbf**
}		
}		
}		
uniqueTdFilterElementCount;	**6**	**uimsbf**
for (k=0; k<uniqueTdFilterElementCount; k++) {		
eqFilterFormat;	**1**	**bslbf**
if (eqFilterFormat == 0) { /* pole / zero */		
bsRealZeroRadiusOneCount;	**3**	**uimsbf**
realZeroCount;	**6**	**uimsbf**
genericZeroCount;	**6**	**uimsbf**
realPoleCount;	**4**	**uimsbf**
complexPoleCount;	**4**	**uimsbf**
realZeroRadiusOneCount = 2* bsRealZeroRadiusOneCount;		
for (m=0; m<realZeroRadiusOneCount; m++) {		
zeroSign;	**1**	**uimsbf**
}		
for (m=0; m<realZeroCount; m++) {		
bsZeroRadius;	**7**	**uimsbf**
zeroSign;	**1**	**uimsbf**
}		
for (m=0; m<genericZeroCount; m++) {		
bsZeroRadius;	**7**	**uimsbf**
bsZeroAngle;	**7**	**uimsbf**
}		
for (m=0; m<realPoleCount; m++) {		
bsPoleRadius;	**7**	**uimsbf**
poleSign;	**1**	**uimsbf**

Syntax	No. of bits	Mnemonic
}		
for (m=0; m<complexPoleCount; m++) {		
bsPoleRadius;	**7**	**uimsbf**
bsPoleAngle;	**7**	**uimsbf**
}		
} else { /* FIR coefficients */		
firFilterOrder;	**7**	**uimsbf**
firSymmetry;	**1**	**bslbf**
for (m=0; m<firFilterOrder/2+1; m++) {		
bsFirCoefficient;	**11**	**uimsbf**
}		
}		
}		
uniqueEqSubbandGainsCount;	**6**	**uimsbf**
if (uniqueEqSubbandGainsCount > 0) {		
eqSubbandGainRepresentation;	**1**	**bslbf**
eqSubbandGainFormat;	**4**	**uimsbf**
switch (eqSubbandGainFormat) {		
case GAINFORMAT_QMF32:		
eqSubbandGainCount = 32;		
break;		
case GAINFORMAT_QMFHYBRID39		
eqSubbandGainCount = 39;		
break;		
case GAINFORMAT_QMF64:		
eqSubbandGainCount = 64;		
break;		
case GAINFORMAT_QMFHYBRID71:		
eqSubbandGainCount = 71		
break;		
case GAINFORMAT_QMF128:		
eqSubbandGainCount = 128		
break;		
case GAINFORMAT_QMFHYBRID135:		
eqSubbandGainCount = 135		
break;		
case GAINFORMAT_UNIFORM:		
case default:		

Syntax	No. of bits	Mnemonic
eqSubbandGainCount = **bsEqGainCount** + 1;	**8**	**uimsbf**
break;		
}		
for (k=0; k< uniqueEqSubbandGainsCount; k++) {		
subbandGainsIndex = k;		
if (eqSubbandGainRepresentation == 1) {		
eqSubbandGainSpline();		
} else {		
for (m=0; m<eqSubbandGainCount; m++) {		
bsEqSubbandGain;	**9**	**uimsbf**
}		
}		
}		
}		
}		

Table 87 — Syntax of in-stream eqSubbandGainSpline() payload

Syntax	No. of bits	Mnemonic
eqSubbandGainSpline()		
{		
nEqNodes = **bsEqNodeCount** + 2;	**5**	**uimsbf**
for (k=0; k<nEqNodes; k++) {		
eqSlopeCode[k];	**1..5**	**vlclbf**
}		
for (k=1; k<nEqNodes; k++) {		
eqFreqDeltaCode[k];	**4**	**uimsbf**
}		
eqGainInitialCode;	**5..7**	**vlclbf**
for (k=1; k<nEqNodes; k++) {		
eqGainDeltaCode[k];	**5**	**uimsbf**
}		
}		

Table 88 — Syntax of eqCoefficients() payload for ISO/IEC 14496-12

aligned(8) class EQCoefficients extends FullBox('ueqc', version=0, flags=0) {
bit(1) reserved = 0;
bit(1) eq_delay_max_present;
if (eq_delay_max_present==1) {

```
        bit(8)    bs_eq_delay_max;
    }
    unsigned int(6)    unique_filter_block_count;
    for (j=1; j<=unique_filter_block_count; j++) {
        bit(2)              reserved = 0;
        unsigned int(6)    filter_element_count;
        for (k=1; k<=filter_element_count; k++) {
            bit(1)              reserved = 0;
            unsigned int(6)    filter_element_index;
            bit(1)              bs_filter_element_gain_present;
            if (bs_filter_element_gain_present == 1) {
                bit(6)              reserved = 0;
                unsigned int(10)    bs_filter_element_gain;
            }
        }
    }
    bit(2)              reserved = 0;
    unsigned int(6)    unique_td_filter_element_count;
    for (j=1; j<=unique_td_filter_element_count; j++) {
        bit(7) reserved = 0;
        bit(1) EQ_filter_format;
        if (EQ_filter_format == 0) {
            bit(1)              reserved = 0;
            unsigned int(3)    bs_real_zero_radius_one_count;
            unsigned int(6)    real_zero_count;
            unsigned int(6)    complex_zero_count;
            unsigned int(4)    real_pole_count;
            unsigned int(4)    complex_pole_count;
            for (m=1; m<=bs_real_zero_radius_one_count; m++) {
                unsigned int(1)    zero_sign;
            }
            bit(byte_align(bs_real_zero_radius_one_count))    reserved = 0;       // byte fill
            for (m=1; m<=real_zero_count; m++) {
                unsigned int(7)    bs_zero_radius;
                unsigned int(1)    zero_sign;
            }
            for (m=1; m<=complex_zero_count; m++) {
                bit(2)              reserved = 0;
                unsigned int(7)    bs_zero_radius;
```

```
                unsigned int(7)     bs_zero_angle;
            }
            for (m=1; m<=real_pole_count; m++) {
                unsigned int(7)     bs_pole_radius;
                unsigned int(1)     zero_sign;
            }
            for (m=1; m<=complex_pole_count; m++) {
                bit(2)              reserved = 0;
                unsigned int(7)     bs_pole_radius;
                unsigned int(7)     bs_pole_angle;
            }
        } else {                    // FIR coefficients
            unsigned int(7)     FIR_filter_order;
            bit(1)              FIR_symmetry;
            bit(byte_align(11*(FIR_filter_order/2+1))) reserved = 0;        // byte fill
            for (m=1; m<=FIR_filter_order/2+1; m++) {
                unsigned int(11) bs_FIR_coefficients;
            }
        }
    }
    bit(2)              reserved = 0;
    unsigned int(6)     unique_EQ_subband_gains_count;
    if (unique_EQ_subband_gains_count > 0) {
        bit(3)              reserved = 0;
        bit(1)              EQ_subband_gain_repesentation;
        unsigned int(4)     EQ_subband_gain_format;
        if (EQ_subband_gain_format== GAINFORMAT_UNIFORM) {
            bit(2)              reserved = 0;
            unsigned int(6)     bs_EQ_gain_count;
        }
        for (k=1; k<=unique_subband_gains_count; k++) {
            if (EQ_subband_gain_repesentation==1) {
                unsigned int(5)     bs_EQ_node_count;
                unsigned int        n_EQ_nodes = bs_EQ_node_count + 2;
                unsigned int        bitsize = 5;
                for (k=1; k<=n_EQ_nodes; k++) {
                    unsigned int(1...5)     EQ_slope_code;
                    bitsize = bitsize + sizeof(EQ_slope_code);
                }
```

```
                bit(byte_align(bitsize))    reserved = 0;                    // fill to byte boundary
                for (k=2; k<=n_EQ_nodes; k++) {
                    unsigned int(4)    EQ_freq_delta_code;
                }
                bit(byte_align(4 * (n_EQ_nodes-1))) reserved = 0;            // fill to byte boundary
                unsigned int(5...7) EQ_gain_initial_code;
                bit(8 - sizeof(EQ_gain_initial_code)) reserved = 0;      // byte fill
                for (k=2; k<=n_EQ_nodes; k++) {
                    unsigned int(5) EQ_gain_delta_code;
                }
                bit(byte_align(5 * (n_EQ_nodes-1))) reserved = 0;            // byte fill
            } else {                        // EQ_subband_gain_count derived as in Table 86
                for (m=1; m<=EQ_subband_gain_count; m++) {
                    unsigned int(9) bs_EQ_subband_gain;
                }
                bit(byte_align(9*EQ_subband_gain_count))    reserved = 0;    // byte fill
            }
        }
    }
}
```

Table 89 — Syntax of in-stream eqInstructions() payload

Syntax	No. of bits	Mnemonic
eqInstructions()		
{		
eqSetId;	**6**	**uimsbf**
eqSetComplexityLevel;	**4**	**uimsbf**
channelCount = baseChannelCount;		
downmixIdPresent;	**1**	**bslbf**
if (downmixIdPresent==1) {		
downmixId;	**7**	**uimsbf**
eqApplyToDownmix;	**1**	**bslbf**
additionalDownmixIdPresent;	**1**	**bslbf**
if (additionalDdownmixIdPresent==1) {		
additionalDownmixIdCount;	**7**	**uimsbf**
for (i=0; i<additionalDownmixIdCount; i++) {		
additionalDownmixId;	**7**	**uimsbf**
}		
} else {		

Syntax	No. of bits	Mnemonic
additionalDownmixIdCount = 0;		
}		
if ((eqApplyToDownmix == 1) && (downmixId != 0) &&		
(downmixId != 0x7F) && (additionalDownmixIdCount==0)) {		
channelCount = targetChannelCountFromDownmix;		
} else if (downmixId == 0x7F) {		
channelCount = 1;		
}		
} else {		
downmixId = 0;		
}		
drcSetId;	**6**	**uimsbf**
additionalDrcSetIdPresent;	**1**	**bslbf**
if (additionalDrcSetIdPresent ==1) {		
additionalDrcSetIdCount;	**6**	**uimsbf**
for (i=0; i<additionalDrcSetIdCount; i++) {		
additionalDrcSetId;	**6**	**uimsbf**
}		
} else {		
additionalDrcSetIdCount = 0;		
}		
eqSetPurpose;	**16**	**bslbf**
dependsOnEqSetPresent;	**1**	**bslbf**
if (dependsOnEqSetPresent ==1) {		
dependsOnEqSet;	**6**	**uimsbf**
} else {		
noIndependentEqUse;	**1**	**bslbf**
}		
eqChannelGroupCount = 0;		
for (i=0; i<channelCount; i++) {		
newGroup = 1;		
eqChannelGroupForChannel[i] = **eqChannelGroup;**	**7**	**uimsbf**
for (k=0; k<i; k++) {		
if (eqChannelGroup == eqChannelGroupForChannel[k]) {		
newGroup = 0;		
}		
}		
if (newGroup == 1) {		

Syntax	No. of bits	Mnemonic
eqChannelGroupCount += 1;		
}		
}		
tdFilterCascadePresent;	**1**	**bslbf**
if (tdFilterCascadePresent == 1) {		
for (i=0; i<eqChannelGroupCount; i++) {		
eqCascadeGainPresent;	**1**	**bslbf**
if (eqCascadeGainPresent==1) {		
bsEqCascadeGain;	**10**	**uimsbf**
}		
filterBlockCount;	**4**	**uimsbf**
for (k=0; k<filterBlockCount; k++) {		
filterBlockIndex;	**7**	**uimsbf**
}		
}		
eqPhaseAlignmentPresent;	**1**	**bslbf**
for (i=0; i<eqChannelGroupCount; i++) {		
for (k=i+1; k<eqChannelGroupCount; k++) {		
if (eqPhaseAlignmentPresent==1) {		
eqPhaseAlignment[i,k] = **bsEqPhaseAlignment;**	**1**	**bslbf**
} else {		
eqPhaseAlignment[i,k] = 1;		
}		
}		
}		
}		
subbandGainsPresent;	**1**	**bslbf**
if (subbandGainsPresent == 1) {		
for (i=0; i<eqChannelGroupCount; i++) {		
subbandGainsIndex;	**6**	**uimsbf**
}		
}		
eqTransitionDurationPresent;	**1**	**bslbf**
if (eqTransitionDurationPresent==1) {		
bsEqTransitionDuration;	**5**	**uimsbf**
}		
}		

Table 90 — Syntax of eqInstructions() payload for ISO/IEC 14496-12

```
aligned(8) class EQInstructions extends FullBox('ueqi', version=0, flags=0) {
    bit(4)              reserved = 0;
    unsigned int (4)    EQ_instructions_count;
    for (a=1; a<=EQ_instructions_count; a++) {
        bit(1)              reserved = 0;
        unsigned int(6)     EQ_set_ID;
        bit(1)              downmix_ID_present;
        if (downmix_ID_present==1) {
            bit(7)              reserved = 0;
            unsigned int(7)     downmix_ID;
            bit(1)              EQ_apply_to_downmix;
            bit(1)              additional_downmix_ID_present;
            if (additional_downmix_ID_present==1) {
                bit(1)              reserved = 0;
                unsigned int(7)     additional_downmix_ID_count;
                for (i=1; i<=additional_downmix_ID_count; i++) {
                    bit(1)              reserved = 0;
                    unsigned int(7)     additional_downmix_ID;
                }
            }
        }
        bit(1)              reserved = 0;
        unsigned int(6)     drc_set_ID;
        bit(1)              additional_drc_set_ID_present;
        if (additional_drc_set_ID_present==1) {
            bit(1)              reserved = 0;
            unsigned int(7)     additional_drc_set_ID_count;
            for (i=1; i<=additional_drc_set_ID_count; i++) {
                bit(2)              reserved = 0;
                unsigned int(6)     additional_drc_set_ID;
            }
        }
        bit(3)    reserved = 0;
        bit(4)    EQ_complexity_level;
        bit(16)    EQ_set_purpose;
        bit(1)    depends_on_EQ_set_present;
        if (depends_on_EQ_set_present==1) {
            bit(2)              reserved = 0;
```

```
        unsigned int(6)     depends_on_EQ_set;
    } else {
        bit(7)     reserved = 0;
        bit(1)     no_independent_EQ_use;
    }
    bit(7)                  reserved = 0;
    unsigned int(7)     channelCount;
    unsigned int(7)     EQ_channel_group_count;
    bit(1)                  td_filter_cascade_present;
    bit(1)                  subband_gains_present;
    bit(1)                  EQ_transition_duration_present;
    if (td_filter_cascade_present == 1) {
        for (i=1; i<=EQ_channel_group_count; i++) {
            bit(3)                  reserved = 0;
            bit(1)                  EQ_cascade_gain_present;
            unsigned int(4)     filter_block_count;
            if (EQ_cascade_gain_present == 1) {
                bit(6)                  reserved = 0;
                unsigned int(10)     bs_EQ_cascade_gain;
            }
            for (k=1; k<=filter_block_count; k++) {
                bit(1)                  reserved = 0;
                unsigned int(7)     filter_block_index;
            }
        }
        bit(7)     reserved = 0;
        bit(1)     EQ_phase_alignment_present;
        if (EQ_phase_alignment_present == 1) {
            unsigned int     bitsize = 0;
            for (i=1; i<=EQ_channel_group_count; i++) {
                for (k=i+1; k<=EQ_channel_group_count; k++) {
                    bit(1)     bs_EQ_phase_alignment;
                    bitsize++;
                }
            }
            bit(byte_align(bitsize))     reserved = 0;     // fill to byte boundary
        }
    }
    if (subband_gains_present == 1) {
```

```
            for (i=1; i<=EQ_channel_group_count; i++) {
                bit(2)              reserved = 0;
                unsigned int(6)     subband_gains_index;
            }
        }
        if (EQ_transition_duration_present == 1) {
            bit(3)              reserved = 0;
            unsigned int(5)     bs_EQ_transition_duration;
        }
    }
}
```

8 Reference software

8.1 Reference software structure

8.1.1 General

Clause 8 contains reference software for MPEG-D DRC which has been derived from reference models used in the process of developing this document. The software is provided at https://standards.iso.org/iso-iec/23003/-4/ed-2/en.

The implementation of this reference software is compliant with the MPEG-D DRC decoding processes described in this document. Complying ISO/IEC 23003-4 implementations are not expected to follow the algorithms or the programming techniques used by the reference software. The decoding software is considered compliant but cannot add anything to the textual technical description of MPEG-D DRC included in this document.

The software described in Clause 8 and in Annex K is divided into three categories:

a) **Bitstream decoding software** is catalogued in 8.2. This software accepts bitstreams encoded according to the normative specification in this document and applies the bitstream content to an input audio file. Attention is drawn to the fact that several different implementations can produce the same result but that the implementation of the software provided is compliant with the MPEG-D DRC decoding processes described in this document.

b) **Bitstream encoding software** is catalogued in K.1. This software creates bitstreams according to the specification in this document. The techniques used for encoding are not specified by this document.

c) **Utility software** is catalogued in K.2. This software was found useful by the developers of this document, but their technical specification is outside of the scope of this document.

The software is provided at http://standards.iso.org/iso-iec/23003/-4/ed-2/en.

8.2 Bitstream decoding software

8.2.1 General

The provided bitstream decoding software is a reference implementation of the specification text in this document.

8.2.2 MPEG-D DRC decoding software

Table 91 — MPEG-D DRC decoding software

Location	Content
MPEG_D_DRC_refsoft\modules\drcTool\drcToolDecoder	MPEG-D DRC Decoder (DRC Tool Decoder)
MPEG_D_DRC_refsoft\modules\uniDrcModules	DRC Tool Sub-Modules
MPEG_D_DRC_refsoft\modules\peakLimiter	Peak Limiter Module

9 Conformance

9.1 General

Clause 9 specifies conformance criteria for both bitstreams and decoders compliant with the MPEG-D DRC standard as defined in this document. This is done to assist implementers and to ensure interoperability.

9.2 Conformance testing

9.2.1 Conformance test data and test procedure

To test MPEG-D DRC compliant audio decoders, a package of conformance test data which is described in Figure 14 is available at http://standards.iso.org/iso-iec/23003/-4/ed-2/en.

The package contains a zip archive with test tools, test bitstreams, interface files, reference files, test scripts and a Microsoft® Excel[1] worksheet ("ISO-IEC_23003-4_MPEG_D_DRC_Conformance_Tables.xlsx") which defines all conformance test sequences.

To fulfil a conformance test sequence, a decoder under test shall decode the corresponding conformance test bitstream with the given decoder setting and the given input audio file. The output of a decoder under test shall meet the indicated conformance test criteria with respect to the corresponding reference file. Since the modules of an MPEG-D DRC decoder can be integrated in various different codec environments, some test categories and test sequences require testing the output of decoder sub-modules. The test procedure is summarized in Figure 15.

[1] Excel is the trademark of a product supplied by Microsoft®. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of the product named. Equivalent products may be used if they can be shown to lead to the same results.

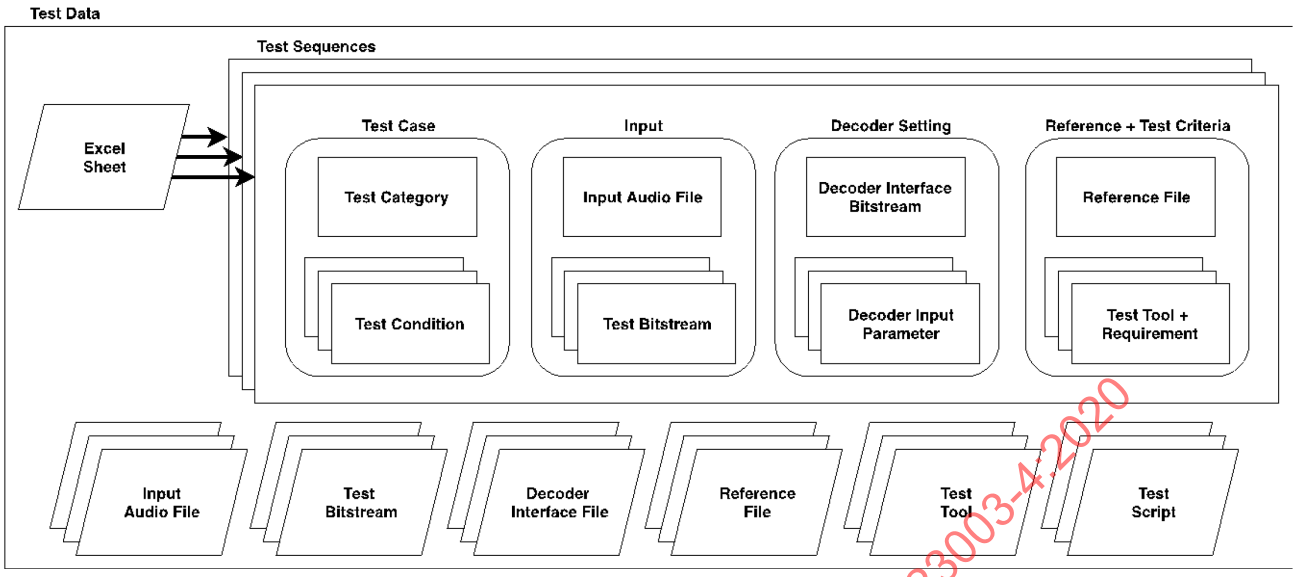


Figure 14 — Conformance test data

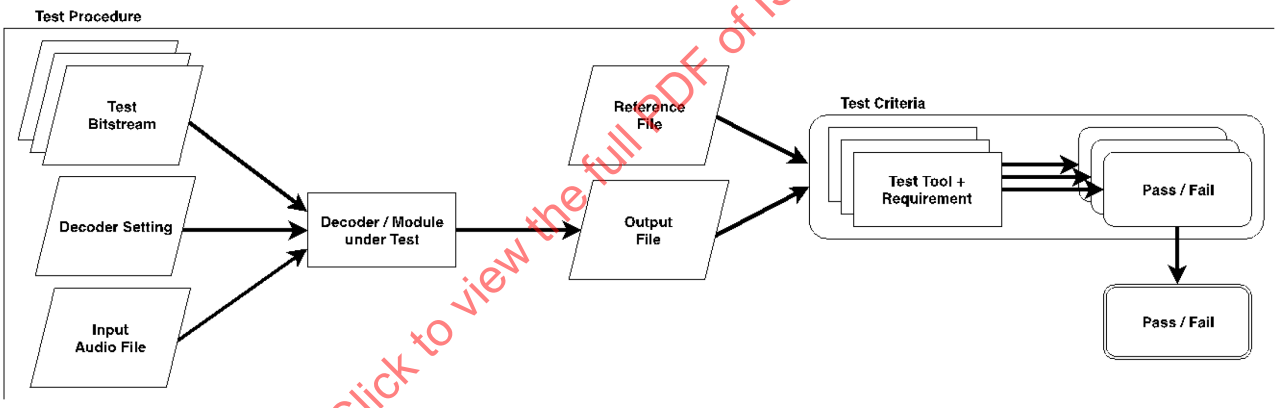


Figure 15 — Conformance test procedure

In cases where the decoder or decoder module under test is followed by additional operations (e.g. quantizing a signal to a 16 bit output signal), the conformance point is prior to such additional operations, i.e. it is permitted to use the actual output (e.g. with more than 16 bit) for conformance testing. Measurements are carried out relative to full scale where the output signals of the decoders are usually normalized to be in the range between -1.0 and +1.0.

Conformance test criteria define which conformance test tools and parameters shall be used to compare reference files with output files from the decoder or decoder module under test. The conformance test tools are defined in subclause 9.2.4.

The characteristics of a conformance test bitstream are defined by the corresponding conformance test case. A conformance test case is defined by a conformance test category and a combination of one or more conformance test conditions if applicable. A definition of all conformance test categories and conformance test conditions can be found in subclause 9.2.4. A conformance test bitstream may be part of several conformance test sequences.

A decoder setting is a combination of a decoder interface bitstream and decoder input parameters that need to be supplied to the MPEG-D DRC decoder for a certain conformance test sequence.

9.2.2 Naming conventions

For all conformance test sequences, the file names are composed of several parts which convey information about:

— which input audio file applies to a conformance test sequence;

— which syntax type applies to a bitstream;

— which test case applies to a conformance test bitstream;

— which decoder setting applies to a conformance test sequence.

The file naming convention is given in Table 92. Strings in box brackets are optional.

Table 92 — File name conventions

Type	File Name
Input audio filename	<audioName>[_<cplx>]
Bitstream or input parameter filename	<baseName>-<structure>
Reference filename	<baseName>__<testCat>[-<decSetting>][__<testCond>][__<audioName>][_<cplx>]

<audioName> Input audio name: “audio-<nCh>ch-<fs>fs-<bit>bit-<desc>-[<domain>]”.

<nCh> Number of channels.

<fs> Sampling rate in Hz.

<bit> Bits per sample, where “16”/”24” is PCM integer and “32” is IEEE floating point format.

<desc> Description string.

<domain> Sub-band domain identifier as listed in Table 93.

<cplx> Complex format string: “real” or “imag”.

<baseName> Base name for bitstreams and input parameter sets: “<prefix>-<syntax>-<N>”.

<prefix> Prefix: “if” for interface bitstreams and interface parameter sets, “bs” for bitstreams and bitstream parameter sets, “sl” for DRC set selection parameter files.

<syntax> Syntax type string as listed in Table 94.

<N> An enumeration number, unique for each <syntax> type.

<structure> syntax structure string (“uniDrc”, “uniDrcConfig”, “uniDrcGain”, “loudnessInfoSet” or “uniDrcInterface”). If <syntax> is “h0”, “uniDrcConfig”, “loudnessInfoSet” and “uniDrcInterface” are interpreted as “mpegh3daUniDrcConfig”, “mpegh3daLoudnessInfoSet” and “mpegh3daLoudnessDrcInterface”, respectively. File labels “downmixMatrixSet” and “mpegh3daParameters” indicate parameter files as defined in subclause 9.2.3.2.

<testCat> Test category string. Abbreviations as listed in Table 95.

<decSetting> Decoder setting name: “[<baseName>][-id<id>][-<afs>afs][-ld][-<gd>gd][-<ad>ad][-<other>][-<domain>]”.

<id> Gain decoder instance ID that defines whether a selected DRC set shall be applied dependent on its defined *downmixId* value:

- "0": Gain decoder instance accepts DRC sets with *downmixId* = 0x0
- "1": Gain decoder instance accepts DRC sets with *downmixId* = 0x0 || downmixId = 0x7F
- "2": Gain decoder instance accepts DRC sets with *downmixId* = 0x7F
- "3": Gain decoder instance accepts DRC sets with *downmixId* != 0 && *downmixId* != 0x7F
- "4": Gain decoder instance accepts DRC sets with *downmixId* != 0

<afs> Processing frame size in samples. Default: 1024.

ld Low-delay mode indicator.

<gd> Gain delay in samples.

<ad> Audio delay in samples.

<other> Other decoder input parameters.

<testCond> Test condition string. May consist of a concatenation of one or more test condition abbreviations as listed in Table 96.

Table 93 — Sub-band domain identifier

Sub-band domain	<domain>
64-band QMF domain	Qmf64
256-band STFT domain	Stft256

Table 94 — Syntax identifier

Specification	<syntax>
ISO/IEC 23003-4:2015	v0
ISO/IEC 23003-4:2019	v1
ISO/IEC 23008-3:2019	h0

Table 95 — Test categories and abbreviations

Test category	Abbrev.
DRC set selection process test category	DrcSelProc
DRC gain decoder test category	DrcGainDec
DRC tool decoder test category	DrcToolDec
Peak limiter test category	PeakLim

Table 96 — Test conditions and abbreviations

Test condition	Abbrev.
Loudness normalization test condition	Ln
Gain coding and gain interpolation test condition	Gc
Gain modification test condition	Gm
Node reservoir test condition	Nr

Multi-band DRC test condition	Mb
Shaping filter test condition	Sf
Parametric DRC test condition	Pd
Equalization filter test condition	Eq
Loudness equalization support test condition	Le
Multiple DRC sets at one location test condition	Md
Downmix test condition	Dx
Ducking and Fading test condition	Df
Peak limiter test condition	Pl
Handling of extensions test condition	Ex

9.2.3 File format definitions

9.2.3.1 DRC set selection parameters

The result of the DRC set selection process as specified in subclause 6.3 shall be stored as a text file that includes the following parameters:

— number of selected DRC sets (*numSelectedDrcSets*) (one line in text file);

— *drcSetId* and *downmixId* for each selected DRC set (*numSelectedDrcSets* lines in text file);

— loudness normalization gain in dB (one line in text file);

— output peak level in dB (one line in text file);

— selected *boost*, *compress* and *drcCharacteristicTarget* parameters (one line in text file);

— channel count before and after downmix (one line in text file);

— *loudnessEqSetId* and *mixingLevel* for selected loudness EQ (one line in text file) if applicable;

— *eqSetId* for selected EQ before and after downmix (one line in text file) if applicable.

An example file is shown in Table 97.

Table 97 — Example file content for DRC set selection parameters

File content	Interpretation
3	numSelectedDrcSets = 3
1 0	drcSetId[0] = 1, downmixId[0] = 0
3 127	drcSetId[1] = 3, downmixId[1] = 127
4 2	drcSetId[2] = 4, downmixId[2] = 2
5.0	loudnessNormalizationGainDb = 5
-3.5	outputPeakLevelDb = -3.5
1.0 1.0 0	Boost = 1.0, compress = 1.0, drcCharacteristicTarget = 0
6 2	baseChannelCount = 6, targetChannelCount = 2
-1 85.0	loudEqId = -1, mixingLevel = 85.0
0 1	eqSetId[0] = 0, eqSetId[1] = 1

9.2.3.2 MPEG-H 3DA parameters

9.2.3.2.1 Downmix parameters

The content of the downmixMatrixSet() structure according to ISO/IEC 23008-3:2019, Table 27, shall be stored as a text file that includes the following parameters:

— number of *downmixId* definitions (*downmixIdCount*) (one line in text file);

— *downmixId* and *targetChannelCount* for each *downmixId* definition (*downmixIdCount* lines in text file).

An example file is shown is Table 98.

Table 98 — Example file content for downmixMatrixSet parameters

File content	Interpretation
3	downmixIdCount = 3
1 2	downmixId[0] = 1, targetChannelCount[0] = 2
3 6	downmixId[1] = 3, targetChannelCount[1] = 6
4 8	downmixId[2] = 4, targetChannelCount[2] = 8

9.2.3.2.2 Interactivity parameters

User interactivity parameters according to ISO/IEC 23008-3 shall be stored as a text file that includes the following parameters:

— number of active groups for the selected audio scene (*numGroupIdRequests*) (one line in text file);

— *mae_groupID* value for each active group (*numGroupIdRequests* lines in text file);

— number of active group presets for the selected audio scene (*numGroupPresetIDRequests*) (one line in text file);

— *mae_groupPresetID* and number of members for each active group preset (*numGroupPresetIDRequests* lines in text file);

— *mae_groupPresetID* value for a prioritized group preset (default: "-1") (one line in text file).

An example file is shown in Table 99.

Table 99 — Example file content for mpegh3daParameters parameters

File content	Interpretation
3	numGroupIDRequests = 3
1	mae_groupID[0] = 1
3	mae_groupID[1] = 3
4	mae_groupID[2] = 4
1	numGroupPresetIDRequests = 1
2 1	mae_groupPresetID[0] = 2, numMembersGroupPresetID[0] = 1
2	groupPresetIdRequestedPreference = 2

9.2.4 Conformance test tools

9.2.4.1 RMS/LSB measurement

The RMS/LSB measurement shall be as defined in ISO/IEC 14496-26:2010, 7.1.2.2.1.

9.2.4.2 Segmental SNR

The segmental SNR shall be as defined in ISO/IEC 14496-26:2010, 7.1.2.2.2.

9.2.4.3 Conformance of DRC set selection parameters

To fulfil the "Conformance of DRC set selection parameters" test, the DRC set selection parameter result of a decoder under test shall have the following properties with respect to a supplied reference file as defined in subclause 9.2.3.1:

— match of *drcSetId* and *downmixId* for each selected DRC set;

— match of loudness normalization gain with a tolerance of +/- 0.05 dB;

— match of *loudnessEqSetId* and *mixingLevel* for selected loudness EQ if applicable;

— match of *eqSetId* for selected EQ before and after downmix if applicable.

The uniDrcSelectionProcessConformanceCmdl tool is provided to display selection results and compare them with reference results.

9.2.4.4 Profiles/Levels conformance

The uniDrcBitstreamConformanceCmdl tool is supplied to check the bitstream conformance with certain Profiles and Levels. The conformance of self-supplied payloads for DRC configurations, uniDrcConfig(), and loudness information, loudnessInfoSet(), should be tested with the test tool uniDrcBitstreamConformanceCmdl. For MPEG-H 3DA bitstreams, the mpegh3daUniDrcBitstreamConformanceCmdl tool should be used instead. Each tool checks the conformance in terms of

— memory and complexity limits;

— delay limits;

— presence of required metadata elements.

The tools display the metadata of the supplied payloads and report the conformance result. A bitstream is not conformant if one or more of its payloads do not pass the test tool check.

9.2.4.5 Further tool support

The uniDrcInterfaceConformance tool is provided to display the content of interface bitstreams.

9.3 Encoder conformance for MPEG-D DRC bitstreams

9.3.1 Characteristics and test procedure

Characteristics of MPEG-D DRC bitstreams specify the necessary constraints that an encoder shall comply with in order to generate conformant bitstreams. These syntactic and semantic constraints may, for example, restrict the value range of certain parameters or the maximum count of certain parameter sets in the bitstream or require certain metadata to be present.

Each MPEG-D DRC bitstream shall meet the syntactic and semantic requirements specified in this document. Subclause 9.3 defines the conformance criteria that shall be fulfilled by a compliant bitstream dependent on its syntax type and an associated profile and level if applicable. These criteria

are specified for the syntactic elements of the bitstream and for some parameters derived from the bitstream payload.

For each bitstream structure, a set of semantic tests to be performed is described. To verify whether the syntax is correct is straightforward and therefore not defined herein after. It is assumed that the tested bitstreams contain no errors due to transmission or other reasons.

9.3.2 Configuration payload

9.3.2.1 Characteristics

Encoders may apply restrictions to the following parameters of the uniDrcConfig() and loudnessInfoSet() structure defined in Annex A:

a) presence and value of *bsSampleRate*, *bsTimeDeltaMin* or *bsDrcFrameSize*;

b) presence and type of extension elements:

   1) uniDrcConfigExtension();

   2) loudnessInfoSetExtension();

c) restrictions on the number of metadata structures:

   1) *downmixInstructionsCount*, *downmixInstructionsV1Count*;

   2) *drcCoefficientsBasicCount*, *drcCoefficientsUniDrcCount*, *drcCoefficientsUniDrcV1Count*;

   3) *drcInstructionsBasicCount*, *drcInstructionsUniDrcCount*, *drcInstructionsUniDrcV1Count*;

   4) *loudnessInfoCount*, *loudnessInfoV1Count*;

   5) *loudnessInfoAlbumCount*, *loudnessInfoV1AlbumCount*;

   6) *parametricDrcInstructionsCount*;

   7) *loudnessEqInstructionsCount*;

   8) *eqInstructionsCount*;

d) restrictions on the number of metadata parameters:

   1) *characteristicLeftCount*, *characteristicRightCount*, *shapeFilterCount*, *gainSequenceCount*, *gainSetCount*, *bandCount*;

   2) *parametricDrcGainSetCount*;

   3) *uniqueFilterBlockCount*, *uniqueTdFilterElementCount*, *uniqueEqSubbandGainsCount*.

9.3.2.2 Requirements for configuration syntax

9.3.2.2.1 uniDrc()

uniDrcLoudnessInfoSetPresent Shall be set to one in the first frame and for configuration changes if a loudnessInfoSet() or uniDrcConfig() structure is not conveyed by the ISO base media file format (ISO/IEC 14496-12) syntax. Should be set to one for random access points in streaming scenarios as described in subclause 6.11.

uniDrcConfigPresent Shall be set to one in first frame and for configuration changes if an uniDrcConfig() structure is not conveyed by the ISO base media file format (ISO/IEC 14496-12) syntax. Should be set to one for random access points in streaming scenarios as described in subclause 6.11.

9.3.2.2.2 uniDrcConfig()

sampleRatePresent No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5. Further restrictions can apply dependent on the specific codec environment.

bsSampleRate No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5. Further restrictions can apply dependent on the specific codec environment.

downmixInstructionsCount No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.

drcDescriptionsBasicPresent No restrictions apply.

drcCoefficientsBasicCount No restrictions apply.

drcInstructionsBasicCount No restrictions apply.

drcCoefficientsUniDrcCount No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.

drcInstructionsUniDrcCount No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.

uniDrcConfigExtPresent No restrictions apply.

9.3.2.2.3 loudnessInfoSet()

loudnessInfoAlbumCount No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.

loudnessInfoCount No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.

loudnessInfoSetExtPresent No restrictions apply.

9.3.2.2.4 loudnessInfo() and loudnessInfoV1()

NOTE Bitstream elements marked with an asterisk are part of the loudnessInfoV1() payload.

drcSetId No restrictions apply. If the value is not equal to 0x0 or 0x3F, the *drcSetId* value of a present drcInstructionsUniDrc() or a drcInstructionsUniDrcV1() or a drcInstructionsBasic() payload shall match.

eqSetId* No restrictions apply. If the value is not equal to 0x0 or 0x3F, the *eqSetId* value of a present eqInstructions() payload shall match.

downmixId No restrictions apply. If the value is not equal to 0x0 or 0x7F, the value shall match the *downmixId* value of a present downmixInstructions(), downmixInstructionsV1() or downmixMatrixSet() (ISO/IEC 23008-3) payload.

samplePeakLevelPresent No restrictions apply.

bsSamplePeakLevel No restrictions apply.

truePeakLevelPresent No restrictions apply.

bsTruePeakLevel No restrictions apply.

measurementCount No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.

methodDefinition Reserved values as specified in Table A.49 shall be used for decoder testing only (see Table 102). For profile and level dependent requirements, see subclause 9.3.5.

methodValue No restrictions apply.

measurementSystem Reserved values as specified in Table A.50 shall be used for decoder testing only (see Table 102).

reliability No restrictions apply.

9.3.2.2.5 loudnessInfoSetExtension()

NOTE Bitstream elements marked with an asterisk are part of the UNIDRCLOUDEXT_EQ extension.

loudnessInfoSetExtType Reserved values as specified in Table A.11 shall be used for decoder testing only (see Table 102).

bitSizeLen No restrictions apply.

bitSize No restrictions apply.

loudnessInfoV1AlbumCount* No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.

loudnessInfoV1Count* No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.

otherBit No restrictions apply.

9.3.2.2.6 channelLayout()

baseChannelCount Shall not be encoded with a value of 0. For profile and level dependent restrictions, see subclause 9.3.5.

layoutSignalingPresent No restrictions apply.

definedLayout Reserved values according to ISO/IEC 23091-3 shall be used for decoder testing only (see Table 102).

speakerPosition Reserved values according to ISO/IEC 23091-3 shall be used for decoder testing only (see Table 102).

9.3.2.2.7 downmixInstructions() and downmixInstructionsV1()

NOTE Bitstream elements marked with an asterisk are part of the downmixInstructionsV1() payload.

downmixId Shall not be encoded with a value of 0x0 and 0x7F.

targetChannelCount Shall not be encoded with a value of 0. Shall match channel count of *targetLayout* if *targetLayout* is not equal to 0.

targetLayout Reserved values according to ISO/IEC 23091-3 shall be used for decoder testing only (see Table 102).

downmixCoefficientsPresent No restrictions apply.

bsDownmixOffset* Reserved values according to ISO/IEC 14496-12 shall be used for decoder testing only (see Table 102).

bsDownmixCoefficient No restrictions apply.

bsDownmixCoefficientV1* No restrictions apply.

9.3.2.2.8 drcCoefficientsBasic()

drcLocation Reserved values as specified in Table 2 shall be used for decoder testing only (see Table 102). Further restrictions can apply based on the specific codec environment.

drcCharacteristic Reserved values according to ISO/IEC 23091-3 shall be used for decoder testing only (see Table 102).

9.3.2.2.9 drcCoefficientsUniDrc() and drcCoefficientsUniDrcV1()

NOTE Bitstream elements marked with an asterisk are part of the drcCoefficientsUniDrcV1() payload.

drcLocation Reserved values specified in Table 2 shall be used for decoder testing only (see Table 102). For profile and level dependent restrictions, see subclause 9.3.5. Further restrictions can apply based on the specific codec environment.

drcFrameSizePresent No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5. Further restrictions can apply based on the specific codec environment.

bsDrcFrameSize No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5. Further restrictions can apply based on the specific codec environment.

drcCharacteristicLeftPresent* No restrictions apply.

characteristicLeftCount* No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5. A value of 0 is equivalent to *drcCharacteristicLeftPresent* == 0.

characteristicFormat* No restrictions apply.

bsGainLeft* No restrictions apply.

bsIoRatioLeft* No restrictions apply.

bsExpLeft* No restrictions apply.

flipSignLeft* No restrictions apply.

bsCharNodeCount* No restrictions apply.

bsNodeLevelDelta* No restrictions apply.

bsNodeGain* No restrictions apply.

drcCharacteristicRightPresent* No restrictions apply.

characteristicRightCount*	No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5. A value of 0 is equivalent to *drcCharacteristicRightPresent* == 0.
bsGainRight*	No restrictions apply.
bsIoRatioRight*	No restrictions apply.
bsExpRight*	No restrictions apply.
flipSignRight*	No restrictions apply.
shapeFiltersPresent*	No restrictions apply.
shapeFilterCount*	No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5. A value of 0 is equivalent to *shapeFiltersPresent* == 0.
lfCutFilterPresent*	No restrictions apply.
lfCornerFreqIndex*	Reserved values as specified in Table A.35 shall be used for decoder testing only (see Table 102).
lfFilterStrengthIndex*	Reserved values as specified in Table A.33 shall be used for decoder testing only (see Table 102).
lfBoostFilterPresent*	No restrictions apply.
hfCutFilterPresent*	No restrictions apply.
hfCornerFreqIndex*	Reserved values as specified in Table A.36 shall be used for decoder testing only (see Table 102).
hfFilterStrengthIndex*	Reserved values as specified in Table A.34 shall be used for decoder testing only (see Table 102).
hfBoostFilterPresent*	No restrictions apply.
gainSequenceCount*	No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.
gainSetCount	Shall not be encoded with a value of 0. For profile and level dependent restrictions, see subclause 9.3.5.
gainCodingProfile	For drcCoefficientsUniDrcV1(), the value of this field shall be identical for all DRC gain sets that share one or more DRC gain sequences.
gainInterpolationType	For drcCoefficientsUniDrcV1(), the value of this field shall be identical for all DRC gain sets that share one or more DRC gain sequences. For profile and level dependent restrictions, see subclause 9.3.5.
fullFrame	For low-delay mode (*delayMode*==1), the value of this field shall be set to 1. For drcCoefficientsUniDrcV1(), the value of this field shall be identical for all DRC gain sets that share one or more DRC gain sequences.
timeAlignment	For low-delay mode (*delayMode*==1), the value of this field shall be set to 0. For drcCoefficientsUniDrcV1(), the value of this field shall be identical for all DRC gain sets that share one or more DRC gain sequences.

timeDeltaMinPresent For drcCoefficientsUniDrcV1(), the value of this field shall be identical for all DRC gain sets that share one or more DRC gain sequences. For further profile and level dependent restrictions, see subclause 9.3.5. Further restrictions can apply based on the specific codec environment.

bsTimeDeltaMin For drcCoefficientsUniDrcV1(), the value of this field shall be identical for all DRC gain sets that share one or more DRC gain sequences. For further profile and level dependent restrictions, see subclause 9.3.5. Further restrictions can apply based on the specific codec environment.

bandCount Shall not be encoded with a value of 0. For further profile and level dependent restrictions, see subclause 9.3.5. Dependent on the specific codec environment, further restrictions can apply.

drcBandType Shall be 1 if multi-band DRC gains are applied in the time-domain by using the multi-band DRC filter bank as specified in subclause 6.4.12.

indexPresent* No restrictions apply.

bsIndex* The values of the index field shall fulfill the condition *bsIndex* < *gainSequenceCount*.

drcCharacteristicPresent* No restrictions apply.

drcCharacteristicFormatIsCICP* No restrictions apply.

drcCharacteristic Reserved values according to ISO/IEC 23091-3 shall be used for decoder testing only (see Table 102).

drcCharacteristicLeftIndex* Shall be smaller than *drcCharacteristicLeftCount*.

drcCharacteristicRightIndex* Shall be smaller than *drcCharacteristicRightCount*.

crossoverFreqIndex No restrictions apply.

startSubBandIndex Shall map to a sub-band domain index present at the decoder. This mapping is dependent on the specific codec environment.

9.3.2.2.10 drcInstructionsBasic()

drcSetId Shall not be encoded with a value of 0x0 or 0x3F. The *drcSetId* values for all present drcInstructionsUniDrc(), drcInstructionsUniDrcV1() and drcInstructionsBasic() payloads shall be unique for each structure.

drcLocation Shall match the *drcLocation* value of a present drcCoefficientsBasic(), a drcCoefficientsUniDrc() or a drcCoefficientsUniDrcV1() payload.

downmixId No restrictions apply. If the value is not equal to 0x0 or 0x7F, the value shall match the *downmixId* value of a present downmixInstructions() or a downmixInstructionsV1() payload.

additionalDownmixIdPresent No restrictions apply.

additionalDownmixIdCount Shall be larger than zero.

additionalDownmixId No restrictions apply. If the value is not equal to 0x0 or 0x7F, the value shall match the *downmixId* value of a present

	downmixInstructions(), downmixInstructionsV1() or downmixMatrixSet() (ISO/IEC 23008-3) payload.
drcSetEffect	Reserved values as specified in Table A.45 shall be used for decoder testing only (see Table 101).
limiterPeakTargetPresent	No restrictions apply.
bsLimiterPeakTarget	No restrictions apply.
drcSetTargetLoudnessPresent	No restrictions apply.
bsDrcSetTargetLoudnessValueUpper	No restrictions apply.
drcSetTargetLoudnessValueLowerPresent	No restrictions apply.
bsDrcSetTargetLoudnessValueLower	Shall be smaller than *bsDrcSetTargetLoudnessValueUpper*.

9.3.2.2.11 drcInstructionsUniDrc() and drcInstructionsUniDrcV1()

NOTE Bitstream elements marked with an asterisk are part of the drcInstructionsUniDrcV1() payload.

drcSetId	Shall not be encoded with a value of 0x0 or 0x3F. The *drcSetId* values for all present drcInstructionsUniDrc(), drcInstructionsUniDrcV1() and drcInstructionsBasic() payloads shall be unique for each structure.
drcSetComplexityLevel*	Shall be computed and set according to subclause 6.9.2.
drcLocation	Shall match the *drcLocation* value of a present drcCoefficientsBasic(), a drcCoefficientsUniDrc() or a drcCoefficientsUniDrcV1() payload.
downmixIdPresent*	No restrictions apply.
downmixId	No restrictions apply. If the value is not equal to 0x0 or 0x7F, the value shall match the *downmixId* value of a present downmixInstructions(), downmixInstructionsV1() or downmixMatrixSet() (ISO/IEC 23008-3) payload.
drcApplyToDownmix*	No restrictions apply.
additionalDownmixIdPresent	No restrictions apply.
additionalDownmixIdCount	Shall not be encoded with a value of 0.
additionalDownmixId	No restrictions apply. If the value is not equal to 0x0 or 0x7F, the value shall match the *downmixId* value of a present downmixInstructions(), downmixInstructionsV1() or downmixMatrixSet() (ISO/IEC 23008-3) payload.
drcSetEffect	At least one bit shall be non-zero. Reserved bit positions as specified in Table A.45 shall be set for decoder testing only (see Table 102). For profile and level dependent requirements see subclause 9.3.5.
limiterPeakTargetPresent	No restrictions apply.
bsLimiterPeakTarget	No restrictions apply.
drcSetTargetLoudnessPresent	No restrictions apply.

bsDrcSetTargetLoudnessValueUpper	No restrictions apply.
drcSetTargetLoudnessValueLowerPresent	No restrictions apply.
bsDrcSetTargetLoudnessValueLower	Shall be smaller than *bsDrcSetTargetLoudnessValueUpper*.
dependsOnDrcSetPresent	No restrictions apply.
dependsOnDrcSet	The value shall match the *drcSetId* value of a present drcInstructionsUniDrc() or drcInstructionsUniDrcV1() payload.
noIndependentUse	No restrictions apply.
requiresEq*	No restrictions apply.
bsGainSetIndex	The values of this index field shall fulfill the condition *bsGainSetIndex* <= *gainSetCountTotal*, where *gainSetCountTotal* is the sum of *gainSetCount* and *parametricDrcGainSetCount* derived from a present drcCoefficientsBasic(), drcCoefficientsUniDrc(), drcCoefficientsUniDrcV1() and parametricDrcCoefficients() payload with matching *drcLocation* value. Further note that for *drcSetEffect*==“Duck other” only one DRC channel group can define a *bsGainSetIndex* index not equal to 0.
duckingScalingPresent	No restrictions apply.
bsDuckingScaling	No restrictions apply.
repeatParameters	Shall be set to one if *bsGainSetIndex*, *duckingScalingPresent* and *bsDuckingScaling* are repeated for the next channel.
bsRepeatParametersCount	Shall not be encoded with a value larger than the remaining loop count.
repeatGainSetIndex	Shall be set to one if *bsGainSetIndex* is repeated for the next channel.
bsRepeatGainSetCount	Shall not be encoded with a value larger than the remaining loop count.
targetCharacteristicLeftPresent*	No restrictions apply.
targetCharacteristicLeftIndex*	The values of this index field shall fulfill the condition *targetCharacteristicLeftIndex* < *targetCharacteristicLeftCount*, where *targetCharacteristicLeftCount* is defined by a present drcCoefficientsUniDrcV1() payload with matching *drcLocation* value.
targetCharacteristicRightPresent*	No restrictions apply.
targetCharacteristicRightIndex*	The values of this index field shall fulfill the condition *targetCharacteristicRightIndex* < *targetCharacteristicRightCount*, where *targetCharacteristicRightCount* is defined by a present drcCoefficientsUniDrcV1() payload with matching *drcLocation* value.
gainScalingPresent	No restrictions apply.
bsAttenuationScaling	No restrictions apply.
bsAmplificationScaling	No restrictions apply.

gainOffsetPresent	No restrictions apply.
bsGainOffset	No restrictions apply.
shapeFilterPresent*	No restrictions apply.
shapeFilterIndex*	The values of this index field shall fulfill the condition *shapeFilterIndex* < *shapeFilterCount*, where *shapeFilterCount* is defined by a present drcCoefficientsUniDrcV1() payload with matching *drcLocation* value.

9.3.2.2.12 uniDrcConfigExtension()

uniDrcConfigExtType	Reserved types shall only be used for decoder testing (see Table 102).
bitSizeLen	No restrictions apply.
bitSize	No restrictions apply.
parametricDrcInstructionsCount	No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.
downmixInstructionsV1Present	No restrictions apply.
downmixInstructionsV1Count	Shall not be encoded with a value of 0.
drcCoeffsAndInstructionsUniDrcV1Present	No restrictions apply.
drcCoefficientsUniDrcV1Count	No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.
drcInstructionsUniDrcV1Count	No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.
loudEqInstructionsPresent	No restrictions apply.
loudEqInstructionsCount	No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.
eqPresent	No restrictions apply.
eqInstructionsCount	No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.
otherBit	No restrictions apply.

9.3.2.2.13 drcCoefficientsParametricDrc()

drcLocation	No restrictions apply.
parametricDrcFrameSizeFormat	No restrictions apply.
bsParametricDrcFrameSize	No restrictions apply.
bsDrcFrameSize	No restrictions apply.
parametricDrcDelayMaxPresent	No restrictions apply.
bsParametricDrcDelayMax	No restrictions apply.

resetParametricDrc	No restrictions apply.
parametricDrcGainSetCount	No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5
parametricDrcId	Shall match the *parametricDrcId* of one parametricDrcInstructions() payload.
sideChainConfigType	Reserved values shall be applied for decoder testing only (see Table 102).
downmixId	No restrictions apply. If the value is not equal to 0x0 or 0x7F, the value shall match the *downmixId* value of a present downmixInstructions(), downmixInstructionsV1() or downmixMatrixSet() (ISO/IEC 23008-3) payload.
levelEstimChannelWeightFormat	Reserved values shall be applied for decoder testing only (see Table 102).
addChannel	No restrictions apply.
bsChannelWeight	No restrictions apply.
drcInputLoudnessPresent	No restrictions apply.
bsDrcInputLoudness	No restrictions apply.

9.3.2.2.14 parametricDrcInstructions()

parametricDrcId	No restrictions apply. Shall have a unique value for each parametricDrcInstructions() payload.
parametricDrcLookAheadPresent	No restrictions apply.
bsParametricDrcLookAhead	No restrictions apply.
parametricDrcPresetIdPresent	No restrictions apply.
parametricDrcPresetId	Reserved values shall be applied for decoder testing only (see Table 102).
parametricDrcType	Reserved types shall be used for decoder testing only (see Table 101).
bitSizeLen	No restrictions apply.
bitSize	No restrictions apply.
otherBit	No restrictions apply.

9.3.2.2.15 parametricDrcTypeFeedForward()

levelEstimKWeightingType	Reserved values shall be applied for decoder testing only (see Table 102).
levelEstimIntegrationTimePresent	No restrictions apply.
bsLevelEstimIntegrationTime	No restrictions apply.
drcCurveDefinitionType	No restrictions apply.

drcCharacteristic	Reserved values according to ISO/IEC 23091-3 shall be used for decoder testing only (see Table 102).
bsNodeCount	No restrictions apply.
bsNodeLevelInitial	No restrictions apply.
bsNodeLevelDelta	No restrictions apply.
bsNodeGain	No restrictions apply.
drcGainSmoothParametersPresent	No restrictions apply.
bsGainSmoothAttackTimeSlow	No restrictions apply.
bsGainSmoothReleaseTimeSlow	No restrictions apply.
gainSmoothTimeFastPresent	No restrictions apply.
bsGainSmoothAttackTimeFast	No restrictions apply.
bsGainSmoothReleaseTimeFast	No restrictions apply.
gainSmoothThresholdPresent	No restrictions apply.
bsGainSmoothAttackThreshold	No restrictions apply.
bsGainSmoothReleaseThreshold	No restrictions apply.
gainSmoothHoldOffCountPresent	No restrictions apply.
bsGainSmoothHoldOff	No restrictions apply.

9.3.2.2.16 parametricDrcTypeLimiter()

parametricLimThresholdPresent	No restrictions apply.
bsParametricLimThreshold	No restrictions apply.
parametricLimReleaseTimePresent	No restrictions apply.
bsParametricLimReleaseTime	No restrictions apply.

9.3.2.2.17 loudEqInstructions()

loudEqSetId	Shall not be encoded with a value of 0x0 or 0x3F. Shall have a unique value for each loudEqInstructions() payload.
drcLocation	No restrictions apply.
downmixIdPresent	No restrictions apply.
downmixId	No restrictions apply. If the value is not equal to 0x0 or 0x7F, the value shall match the *downmixId* value of a present downmixInstructions(), downmixInstructionsV1() or downmixMatrixSet() (ISO/IEC 23008-3) payload.
additionalDownmixIdPresent	No restrictions apply.
additionalDownmixIdCount	Shall be larger than zero.

additionalDownmixId	No restrictions apply. If the value is not equal to 0x0 or 0x7F, the value shall match the *downmixId* value of a present downmixInstructions(), downmixInstructionsV1() or downmixMatrixSet() (ISO/IEC 23008-3) payload.
drcSetIdPresent	No restrictions apply.
drcSetId	Shall not be encoded with a value of 0x0 or 0x3F. Shall match the *drcSetId* of a drcInstructionsUniDrc(), drcInstructionsUniDrcV1() or drcInstructionsBasic() payload.
additionalDrcSetIdPresent	No restrictions apply.
additionalDrcSetIdCount	Shall be larger than zero.
additionalDrcSetId	Shall not be encoded with a value of 0x0 or 0x3F. Shall match the *drcSetId* of a drcInstructionsUniDrc(), drcInstructionsUniDrcV1() or drcInstructionsBasic() payload.
eqSetIdPresent	No restrictions apply.
eqSetId	Shall not be encoded with a value of 0x0 or 0x3F. Shall match the *eqSetId* of an eqInstructions() payload.
additionalEqSetIdPresent	No restrictions apply.
additionalEqSetIdCount	Shall be larger than zero.
additionalEqSetId	Shall not be encoded with a value of 0x0 or 0x3F. Shall match the *eqSetId* of an eqInstructions() payload.
loudnessAfterDrc	No restrictions apply.
loudnessAfterEq	No restrictions apply.
loudEqGainSequenceCount	Shall not exceed the actual number of gain sequences.
gainSequenceIndex	Shall be smaller than the number of available gain sequences.
drcCharacteristicFormatCICP	No restrictions apply.
drcCharacteristic	Referenced characteristic shall be invertible (see Table A.81). Reserved values shall be used for decoder testing only (see Table 101).
drcCharacteristicLeftIndex	Shall be smaller than *drcCharacteristicLeftCount*. Referenced characteristic shall be invertible.
drcCharacteristicRightIndex	Shall be smaller than *drcCharacteristicRightCount*. Referenced characteristic shall be invertible.
frequencyRangeIndex	Reserved values shall be used for decoder testing only (see Table 101).
bsLoudEqScaling	No restrictions apply.
bsLoudEqOffset	No restrictions apply.

9.3.2.2.18 eqCoefficients()

eqDelayMaxPresent	No restrictions apply.
bsEqDelayMax	No restrictions apply.
uniqueFilterBlockCount	No restrictions apply.
filterElementCount	No restrictions apply.
filterElementIndex	Shall be less than *uniqueTdFilterElementCount* for time domain EQ and shall be less than *uniqueEqSubbandGainsCount* for subband domain EQ.
filterElementGainPresent	No restrictions apply.
bsFilterElementGain	No restrictions apply.
uniqueTdFilterElementCount	No restrictions apply.
eqFilterFormat	No restrictions apply.
bsRealZeroRadiusOneCount	No restrictions apply.
realZeroCount	Shall be zero unless part of filter blocks that contain only a single filter element and if these filter blocks are not subject to cascade phase alignment.
genericZeroCount	No restrictions apply.
realPoleCount	No restrictions apply.
complexPoleCount	No restrictions apply.
zeroSign	No restrictions apply.
bsZeroRadius	No restrictions apply.
bsZeroAngle	No restrictions apply.
bsPoleRadius	Shall not be encoded with a value of 0x0.
poleSign	No restrictions apply.
bsPoleAngle	No restrictions apply.
firFilterOrder	No restrictions apply.
firSymmetry	No restrictions apply.
bsFirCoefficient	No restrictions apply.
uniqueEqSubbandGainsCount	No restrictions apply.
eqSubbandGainRepresentation	No restrictions apply.
eqSubbandGainFormat	Reserved values shall be used for decoder testing only (see Table 101).
bsEqGainCount	No restrictions apply.

9.3.2.2.19 eqSubbandGainSpline()

bsEqNodeCount	No restrictions apply.
eqSlopeCode	No restrictions apply.
eqFreqDeltaCode	No restrictions apply.
eqGainInitialCode	No restrictions apply.
eqGainDeltaCode	No restrictions apply.

9.3.2.2.20 eqInstructions()

eqSetId	Shall not be encoded with a value of 0x0 or 0x3F. Shall take a unique value for each eqInstructions() payload.
eqSetComplexityLevel	Shall be computed and set according to subclause 6.9.3.
downmixIdPresent	No restrictions apply.
downmixId	No restrictions apply. If the value is not equal to 0x0 or 0x7F, the value shall match the *downmixId* value of a present downmixInstructions(), downmixInstructionsV1() or downmixMatrixSet() (ISO/IEC 23008-3) payload.
eqApplyToDownmix	No restrictions apply.
additionalDownmixIdPresent	No restrictions apply.
additionalDownmixIdCount	Shall be larger than zero.
additionalDownmixId	No restrictions apply. If the value is not equal to 0x0 or 0x7F, the value shall match the *downmixId* value of a present downmixInstructions(), downmixInstructionsV1() or downmixMatrixSet() (ISO/IEC 23008-3) payload.
drcSetId	Shall not be encoded with a value of 0x0 or 0x3F. Shall match the *drcSetId* of a drcInstructionsUniDrc(), drcInstructionsUniDrcV1() or drcInstructionsBasic() payload..
additionalDrcSetIdPresent	No restrictions apply.
additionalDrcSetIdCount	Shall be larger than zero.
additionalDrcSetId	Shall not be encoded with a value of 0x0 or 0x3F. Shall match the *drcSetId* of a drcInstructionsUniDrc(), drcInstructionsUniDrcV1() or drcInstructionsBasic() payload.
eqSetPurpose	At least one bit shall be set. Reserved bit positions according to Table A.89 shall be set to one for testing only (see Table 102).
dependsOnEqSetPresent	No restrictions apply.
dependsOnEqSet	If present, shall take a value of a valid *eqSetId* from another eqInstructions() payload.
noIndependentEqUse	No restrictions apply.
eqChannelGroup	Shall be smaller than the actual number of channel groups.

tdFilterCascadePresent	No restrictions apply.
eqCascadeGainPresent	No restrictions apply.
bsEqCascadeGain	No restrictions apply.
filterBlockCount	No restrictions apply.
filterBlockIndex	Shall be smaller than *filterBlockCount*.
eqPhaseAlignmentPresent	No restrictions apply.
bsEqPhaseAlignment	No restrictions apply.
subbandGainsPresent	No restrictions apply.
subbandGainsIndex	Shall be smaller than the number of EQ subbands.
eqTransitionDurationPresent	No restrictions apply.
bsEqTransitionDuration	No restrictions apply.

9.3.3 Interface payload

9.3.3.1 Characteristics

Encoders may apply restrictions to the following parameters of the uniDrcInterface() structure defined in Annex B:

a) presence of uniDrcInterfaceSignature;

b) presence of individual payloads:

1) systemInterface();
2) loudnessNormalizationControlInterface();
3) loudnessNormalizationParameterInterface();
4) dynamicRangeControlInterface();
5) dynamicRangeControlParameterInterface();
6) uniDrcInterfaceExtension();
7) loudnessEqParameterInterface();
8) equalizationControlInterface().

9.3.3.2 Requirements for interface syntax

9.3.3.2.1 uniDrcInterface()

uniDrcInterfaceSignaturePresent	No restrictions apply.
uniDrcInterfaceSignatureType	No restrictions apply.
bsUniDrcInterfaceSignatureDataLength	No restrictions apply.
uniDrcInterfaceSignatureData	No restrictions apply.

systemInterfacePresent	No restrictions apply.
loudnessNormalizationControlInterfacePresent	No restrictions apply.
loudnessNormalizationParameterInterfacePresent	No restrictions apply.
dynamicRangeControlInterfacePresent	No restrictions apply.
dynamicRangeControlParameterInterfacePresent	No restrictions apply.
uniDrcInterfaceExtensionPresent	No restrictions apply.

9.3.3.2.2 systemInterface()

targetConfigRequestType	No restrictions apply.
numDownmixIdRequests	No restrictions apply.
downmixIdRequested	No restrictions apply.
targetLayoutRequested	No restrictions apply.
targetChannelCountRequested	No restrictions apply.

9.3.3.2.3 loudnessNormalizationControlInterface()

loudnessNormalizationOn	No restrictions apply.
targetLoudness	No restrictions apply.

9.3.3.2.4 loudnessNormalizationParameterInterface()

albumMode	No restrictions apply.
peakLimiterPresent	No restrictions apply.
changeLoudnessDeviationMax	No restrictions apply.
loudnessDeviationMax	No restrictions apply.
changeLoudnessMeasurementMethod	No restrictions apply.
loudnessMeasurementMethod	No restrictions apply.
changeLoudnessMeasurementSystem	No restrictions apply.
loudnessMeasurementSystem	No restrictions apply.
changeLoudnessMeasurementPreProc	No restrictions apply.
loudnessMeasurementPreProc	No restrictions apply.
changeDeviceCutOffFrequency	No restrictions apply.
deviceCutOffFrequency	No restrictions apply.
changeLoudnessNormalizationGainDbMax	No restrictions apply.
loudnessNormalizationGainDbMax	No restrictions apply.
changeLoudnessNormalizationGainModificationDb	No restrictions apply.
loudnessNormalizationGainModificationDb	No restrictions apply.

changeOutputPeakLevelMax	No restrictions apply.
outputPeakLevelMax	No restrictions apply.

9.3.3.2.5 dynamicRangeControlInterface()

dynamicRangeControlOn	No restrictions apply.
numDrcFeatureRequests	No restrictions apply.
drcFeatureRequestType	No restrictions apply.
numDrcEffectTypeRequests	No restrictions apply.
numDrcEffectTypeRequestsDesired	No restrictions apply.
drcEffectTypeRequest	No restrictions apply.
dynRangeMeasurementRequestType	No restrictions apply.
dynRangeRequestedIsRange	No restrictions apply.
dynamicRangeRequestValue	No restrictions apply.
dynamicRangeRequestValueMin	No restrictions apply.
dynamicRangeRequestValueMax	No restrictions apply.
drcCharacteristicRequest	No restrictions apply.

9.3.3.2.6 dynamicRangeControlParameterInterface()

changeCompress	No restrictions apply.
changeBoost	No restrictions apply.
compress	No restrictions apply.
boost	No restrictions apply.
changeDrcCharacteristicTarget	No restrictions apply.
drcCharacteristicTarget	No restrictions apply.

9.3.3.2.7 uniDrcInterfaceExtension()

uniDrcInterfaceExtType	No restrictions apply.
bitSizeLen	No restrictions apply.
bitSize	No restrictions apply.
loudnessEqParameterInterfacePresent	No restrictions apply.
equalizationControlInterfacePresent	No restrictions apply.
otherBit	No restrictions apply.

9.3.3.2.8 loudnessEqParameterInterface()

loudnessEqRequestPresent	No restrictions apply.
loudnessEqRequest	No restrictions apply.

sensitivityPresent No restrictions apply.

bsSensitivity No restrictions apply.

playbackGainPresent No restrictions apply.

bsPlaybackGain No restrictions apply.

9.3.3.2.9 equalizationControlInterfacePresent()

eqSetPurpose No restrictions apply.

9.3.4 Frame Payload

9.3.4.1 Characteristics

Encoders may be required to restrict the following parameters of the uniDrcGain() structure defined in Annex A:

a) delayMode (default or low-delay);

b) nNodes (maximum number of encoded gain values in one DRC frame).

9.3.4.2 Requirements for frame syntax

9.3.4.2.1 uniDrcGain()

nDrcGainSequences No restrictions apply. For profile and level dependent restrictions, see subclause 9.3.5.

uniDrcGainExtPresent No restrictions apply.

9.3.4.2.2 uniDrcGainExtension()

uniDrcGainExtType No restrictions apply.

bitSizeLen No restrictions apply.

bitSize No restrictions apply.

otherBit No restrictions apply.

9.3.4.2.3 drcGainSequence()

drcGainCodingMode No restrictions apply.

gainInitialCode No restrictions apply.

endMarker No restrictions apply.

slopeCode No restrictions apply.

frameEndFlag No restrictions apply.

bitSize No restrictions apply.

timeDeltaCode No restrictions apply.

timeDeltaCode No restrictions apply.

gainInitialCode No restrictions apply.

gainDeltaCode No restrictions apply.

9.3.5 Requirements depending on profiles and levels

Depending on a given profile and level requirement, further restrictions and requirements may apply.

9.3.5.1 MPEG-D DRC - Loudness Control profile

Tools are required to support the Loudness Control profile as indicated in Table I.1. Bitstream payloads are required to support the Loudness Control profile payloads as indicated in Table I.2. For bitstream testing regarding mandatory metadata, the required metadata is specified in Table I.5. Information on conformance test tools is given in subclause 9.2.4.

9.3.5.2 MPEG-D DRC - Dynamic Range Control profile

Tools are required to support the Dynamic Range Control profile as indicated in Table I.1. Bitstream payloads are required to the support the Dynamic Range Control profile payloads as indicated in Table I.2. For bitstream testing regarding mandatory metadata, the required metadata is specified in Table I.5. Information on conformance test tools is given in subclause 9.2.4.

9.3.5.3 MPEG-H 3DA - Low Complexity profile

9.3.5.3.1 General

Requirements on MPEG-D DRC when employed as part of the MPEG-H 3DA Low Complexity Profile are specified in ISO/IEC 23008-3:2019, subclause 4.8.2.2. In the following subclauses, MPEG-D DRC syntax related requirements are listed if independent of the employed decoder level. MPEG-H 3DA syntax related requirements and decoder level dependencies are as specified in ISO/IEC 23008-9.

9.3.5.3.2 drcCoefficientsUniDrc()

drcLocation Shall be set to 1.

drcFrameSizePresent Shall be set to 0.

timeAlignment Shall be set to 0.

timeDeltaMinPresent Shall be set to 0.

gainInterpolationType Shall be set to 1.

startSubBandIndex Shall not exceed a value of 255.

9.3.5.3.3 drcInstructionsUniDrc()

drcLocation Shall be set to 1.

dependsOnDrcSetPresent Shall be set to 0 if *downmixId* is set to 0x0.

bsSequenceIndex Shall be unique in simultaneously applied DRC sets except for *bsSequenceIndex* == 0. *bsSequenceIndex* shall not reference a DRC gain set with *bandCount* larger than 1 if *downmixId* is not set to 0x0.

nDrcChannelGroups Shall be restricted to a maximum value of 1 if *downmixId* is not set to 0x0, where *nDrcChannelGroups* is the number of DRC channel groups as derived in Table 15.

9.3.5.3.4 drcGainSequence()

nNodes Shall be restricted to a maximum value of 32, where *nNodes* is the number of encoded gain values in the current DRC frame as derived in Table 77.

9.4 Decoder conformance test categories and conditions

9.4.1 General

This clause describes a set of test categories and test conditions that shall be applied to verify that a given MPEG-D DRC implementation conforms to this document. Test categories and test conditions are designed such that each tool can be tested individually, thus, the constraints for the corresponding conformance test sequences are set accordingly. However, some tools show interactions and dependencies. To cover that fact, test cases can be composed of one or more test conditions.

9.4.2 Conformance test categories

9.4.2.1 DRC set selection process (DrcSelProc)

The DRC set selection process test category is designed for conformance testing of the DRC set selection as specified in subclause 6.3. The reference output of the DRC set selection is stored as defined in subclause 9.2.3.1. For test cases of this test category, no additional test conditions have to be specified.

9.4.2.2 DRC gain decoder (DrcGainDec)

The DRC gain decoder test category is designed for conformance testing of a single DRC gain decoder instance which includes the application of multiple DRC and EQ sets at one application location. The decoder setting for this test category includes the input DRC set selection parameters as defined in subclause 9.2.3.1.

9.4.2.3 DRC tool decoder (DrcToolDec)

The DRC tool decoder test category is designed for conformance testing of a fully integrated MPEG-D DRC decoder as illustrated in the examples of Figure 1 and Figure 2.

9.4.2.4 Peak limiter (PeakLim)

The peak limiter test category is designed for conformance testing of the peak limiter module as defined in Annex G.

9.4.3 Conformance test conditions

9.4.3.1 Loudness normalization test condition (Ln)

9.4.3.1.1 General

This test condition shall be applied to verify the proper behaviour of the loudness normalization tool and the correct signalling of target loudness and related decoder parameters.

9.4.3.1.2 Test sequences

Bitstreams of this test condition shall be designed such that:

— Some bitstreams contain only loudnessInfo() payloads that include at least program loudness (*methodDefinition*=1) or anchor loudness (*methodDefinition*=2) but no sample or true peak value. Some bitstreams shall additionally include DRC sets with *limiterPeakTargetPresent*=1 or *drcSetTargetLoudnessPresent*=1.

— Some bitstreams contain only loudnessInfo() payloads that include at least program loudness (*methodDefinition*=1) or anchor loudness (*methodDefinition*=2) and sample or true peak value. Some bitstreams shall additionally include DRC sets with *limiterPeakTargetPresent*=1 or *drcSetTargetLoudnessPresent*=1.

— Some bitstreams contain loudnessInfo() payloads for the output signal with and without DRC applied.

— Some bitstreams contain loudnessInfo() payloads for the output signal with and without downmix applied.

— Some bitstreams contain no loudnessInfo() payloads.

— Some bitstreams contain track loudness information with program loudness (*methodDefinition*=1) but no album loudness.

— Some bitstreams contain album loudness information with program loudness (*methodDefinition*=1) but no track loudness.

— Some bitstreams contain track and album loudness information with program loudness (*methodDefinition*=1).

Interface bitstreams shall be designed such that:

— An interface bitstream requests a *loudnessNormalizationGainModificationDb* value that is different from 0 dB and a *targetLoudness* request that results in a non-zero normalization gain in dB.

— An interface bitstream requests a *loudnessNormalizationGainDbMax* value that is smaller than infinity and is actively applied by the DRC tool based on a high *targetLoudness* request.

— Interface bitstreams request all available values of *albumMode* and *peakLimiterPresent*.

— A value for *loudnessDeviationMax* is set that actively limits the loudness normalization gain.

— A non-zero value for *outputPeakLevelMax* is set that actively limits the loudness normalization gain.

Test sequences shall be designed such that different normalization gains are applied by the DRC tool dependent on the bitstream configuration and the referenced decoder setting. For each decoder setting, all bitstream configurations shall be tested that contain relevant parameters for the setting.

9.4.3.2 Gain coding and gain interpolation test condition (Gc)

9.4.3.2.1 General

This test condition shall be applied to verify the proper behaviour of the gain coding and gain interpolation tool and the correct signalling of gain, slope and timing parameters.

9.4.3.2.2 Test sequences

Bitstreams of this test condition shall be designed such that:

— Some bitstreams contain DRC gain sequences with spline interpolation (*gainInterpolationType*=0). All gain, slope and time difference words for each defined *gainCodingProfile* shall be triggered at least once over all bitstreams.

— Some bitstreams contain DRC gain sequences with linear interpolation (*gainInterpolationType*=1). All gain, slope and time difference words for each defined *gainCodingProfile* shall be triggered at least once over all bitstreams.

— Some bitstream contain non-default parameters for *sampleRate*, *timeDeltaMin* and *drcFrameSize*.

Test sequences shall be designed such that all described DRC gain sequences are applied by the DRC tool dependent on the referenced decoder setting.

9.4.3.3 Gain modification test condition (Gm)

9.4.3.3.1 General

This test condition shall be applied to verify the proper behaviour of the gain modification tools and the correct signalling of encoder and decoder supplied modification parameters.

9.4.3.3.2 Test sequences

Bitstreams of this test condition shall be designed such that:

— All defined values of *drcCharacteristic* (see ISO/IEC 23091-3) are set at least once across all bitstreams.

— Some bitstreams contain at least one DRC set and no loudness information.

— Some bitstreams contain at least one DRC set with clipping protection gains that have some values below 0 dB.

— Across all bitstreams, values of *attenuationScaling*, *amplificationScaling*, *duckingScaling* and *gainOffset* that are different from 1.0 are present at least once.

— Some bitstreams have a value of *drcCharacteristicLeftPresent*=1 and *drcCharacteristicRightPresent*=1. For each of these characteristics either *characteristicFormat* of 0 and 1 shall occur in a bitstream. Some of these bitstreams shall have a *targetCharacteristicLeftIndex* and *targetCharacteristicRightIndex* value that differs from *drcCharacteristicLeftIndex* and *drcCharacteristicRightIndex* of the applicable gain sequence. The others of these bitstreams shall have a value of *targetCharacteristicLeftPresent*=0 and *targetCharacteristicRightPresent*=0.

Interface bitstreams shall be designed such that:

— Values for *boost* and *compress* are requested that are different from 1.0.

— All defined values of *drcCharacteristicTarget* (see ISO/IEC 23091-3) are requested.

— An interface bitstream requests a *targetLoudness* that results in a non-zero normalization gain in dB to test the scaling of clipping protection gains.

Test sequences shall be designed such that all of the described configurations are applied by the DRC tool dependent on the referenced decoder setting.

9.4.3.4 Node reservoir test condition (Nr)

9.4.3.4.1 General

This test condition shall be applied to verify the proper behaviour of the node reservoir tool.

9.4.3.4.2 Test sequences

Bitstreams of this test condition shall be designed such that gain nodes of the current frame are shifted to the subsequent frame by using the node reservoir mechanism. Test sequences shall be designed such that the correct timing of gain nodes is applied by the DRC tool. This test condition is restricted to the default delay mode (*delayMode*=1).

9.4.3.5 Multi-band DRC test condition (Mb)

9.4.3.5.1 General

This test condition shall be applied to verify the proper behaviour of the gain application tool for the application of multi-band DRC gains in the time and sub-band domain.

9.4.3.5.2 Test sequences

Bitstreams of this test condition shall be designed such that:

— Some bitstreams contain DRC sets that have time-domain multi-band DRC gain sequences (*bandCount*>1, *drcBandType*=1). All possible indices of *crossoverFreqIndex* shall occur at least once across these bitstreams. The DRC band count of 2, 3, and 4 shall each occur at least once.

— Some bitstreams contain DRC sets that have subband-domain multi-band DRC gain sequences (*bandCount*>1). Some of these bitstreams shall use *drcBandType*=1 and include all possible indices of *crossoverFreqIndex* and band counts of 2, 3, and 4. Some of these bitstreams shall use *drcBandType*=0 and include a variety of values for *startSubBandIndex*.

Test sequences shall be designed such that the described DRC sets with different multi-band DRC gain sequence configurations are applied by the DRC tool dependent on the referenced decoder setting.

9.4.3.6 Shaping filter test condition (Sf)

9.4.3.6.1 General

This test condition shall be applied to verify the proper behaviour of the shaping filter.

9.4.3.6.2 Test sequences

Bitstreams of this test condition shall be designed such they contain shaping filters (*shapeFiltersPresent*==1) with different parameter combinations for *lfCutFilterPresent*, *hfCutFilterPresent*, *lfBoostFilterPresent* and *hfBoostFilterPresent* in the drcCoefficientsUniDrcV1() payload. All possible values of the shaping filter parameters *lfCornerFreqIndex*, *hfCornerFreqIndex*, and *hfFilterStrengthIndex* shall occur at least once for each of the filter parts. The drcInstructionsUniDrcV1() payloads shall contain DRC sets which use each of the defined shaping filters.

Test sequences shall be designed such that DRC sets with different shaping filters are applied by the DRC tool dependent on the referenced decoder setting.

9.4.3.7 Parametric DRC test condition (Pd)

9.4.3.7.1 General

This test condition shall be applied to verify the proper behaviour of the parametric DRC tool.

9.4.3.7.2 Test sequences

Bitstreams of this test condition shall be designed such that they contain DRC sets referencing parametric DRC sequences for the generation of DRC gains at the decoder. The referenced parametric DRC configurations as defined by the drcCoefficientsParametricDrc() and parametricDrcInstructions() payloads shall use different parameter combinations for *parametricDrcLookAhead*, *parametricDrcPresetId and parametricDrcType*. Bitstreams shall include custom configurations for the parametricDrcTypeFeedForward() and the parametricDrcTypeLimiter() structures.

Test sequences shall be designed such that DRC sets with different parametric DRC configurations are applied by the DRC tool dependent on the referenced decoder setting. Some DRC sets shall apply regular and parametric DRC sequences to independent DRC channel groups.

9.4.3.8 Equalization filter test condition (Eq)

9.4.3.8.1 General

This test condition shall be applied to verify the proper behaviour of the equalization tool.

9.4.3.8.2 Test sequences

Bitstreams of this test condition shall be designed such that:

— Some bitstreams contain time-domain EQ filter metadata for filters with multiple blocks (*uniqueFilterBlockCount*) and multiple elements (*filterElementCount*), some or all of which have a gain (*bsFilterElementGainPresent*=1). EQ filter metadata shall have multiple time-domain filter elements (*uniqueTdFilterElementCount*) with either format 0 and 1 (*eqFilterFormat*). For *eqFilterFormat*==0, all zero and pole counts are larger than zero (*bsRealZeroRadIsOneCount*, *realZeroCount*, *genericZeroCount*, *realPoleCount*, *complexPoleCount*). For *eqFilterFormat*==1, the maximum filter order (*firFilterOrder*) shall be used and either symmetry (*firSymmetry*) 0 or 1 shall occur.

— Some bitstreams contain time-domain EQ filters where different EQs are assigned to different channel groups and the phase alignment (*eqPhaseAlignmentPresent*=1) is assigned to some of the channel groups.

— Some bitstreams contain subband-domain EQ filters (*uniqueEqSubbandGainsCount*>0) with either representation (*eqSubbandGainRepresentation*) of 0 and 1. A QMF and a uniform subband gain format (*eqSubbandGainFormat*) shall occur in the bitstreams. For the case of a spline representation (*eqSubbandGainRepresentation*==1), the values of *eqSlopeCode*, *eqFreqDeltaCode*, *eqGainInitialCode*, and *eqGainDeltaCode* shall cover a variety of defined codewords (see Table A.98, Table A.99, Table A.100, Table A.101).

— Some bitstreams contain multiple dependent time-domain EQ sets (*dependsOnEqSetPresent*=1), where one is applied before and one after the downmixer. The EQs shall be assigned to different channel groups.

— Some bitstreams contain multiple subband-domain EQ sets (*dependsOnEqSetPresent*=1), where one is applied before and one after the downmixer. The EQs shall be assigned to different channel groups.

— Some bitstream contain EQ metadata but no DRC metadata.

— Some bitstream contain EQ metadata and DRC metadata.

Test sequences shall be designed such that the described EQ filters are applied by the DRC tool dependent on the referenced decoder setting.

9.4.3.9 Loudness equalization support test condition (Le)

9.4.3.9.1 General

This test condition shall be applied to verify the proper behaviour of the loudness equalization support tool of MPEG-D DRC. Since this document specifies only how the loudness equalization metadata is conveyed to the decoder, the application of the metadata is not tested. The test condition verifies that there are no unintended side effects when loudness equalization metadata is present, but the metadata is not applied.

9.4.3.9.2 Test sequences

Bitstreams of this test condition shall be designed such that:

— Some bitstreams with loudEqInstructions() also contain a type of drcInstructions().

— Some bitstreams with loudEqInstructions() also contain one or more loudnessInfoSets() and do not contain any type of drcInstructions().

9.4.3.10 Multiple DRC sets at one location test condition (Md)

9.4.3.10.1 General

This test condition shall be applied to verify the proper application of multiple DRC sets at one application location in time and sub-band domain.

9.4.3.10.2 Test sequences

Test bitstreams of this test condition shall be designed such that:

— Some bitstreams include multiple time-domain DRC sets that consist of a DRC set with a dependent DRC set (*dependsOnDrcSetPresent*=1) and a DRC set with ducking or fading effect (*drcSetEffect*=0x200, 0x400, 0x800). Test sequences shall be designed such that all DRC sets are applied simultaneously at the same location.

— Some bitstreams include multiple subband-domain DRC sets that consist of a DRC set with a dependent DRC set (*dependsOnDrcSetPresent*=1) and a DRC set with ducking or fading effect (*drcSetEffect*=0x200, 0x400, 0x800). Test sequences shall be designed such that all DRC sets are applied simultaneously at the same location.

— Some bitstreams include multiple time-domain DRC sets that consist of a DRC set with a dependent DRC set (*dependsOnDrcSetPresent*=1) and a DRC set with ducking or fading effect (*drcSetEffect*=0x200, 0x400, 0x800). Test sequences shall be designed such that all DRC sets are applied simultaneously at the same location. Except for the ducking or fading DRC set, each DRC set contains both, parametric DRCs and gain sequence based DRCs applied in different channel groups.

— Some bitstreams include multiple subband-domain DRC sets that consist of a DRC set with a dependent DRC set (*dependsOnDrcSetPresent*=1) and a DRC set with ducking or fading effect (*drcSetEffect*=0x200, 0x400, 0x800). Test sequences shall be designed such that all DRC sets are applied simultaneously at the same location. Except for the ducking or fading DRC set, each DRC set contains both, parametric DRCs and gain sequence based DRCs applied in different channel groups.

9.4.3.11 Downmix test condition (Dx)

9.4.3.11.1 General

This test condition shall be applied to verify the proper application of downmix matrices in time and sub-band domain.

9.4.3.11.2 Test sequences

Test bitstreams of this test condition shall be designed such that:

— Some bitstreams contain downmixInstructions() payloads which are applied during decoding.

— Some bitstreams contain downmixInstructionsV1() payloads which are applied during decoding.

— The set of bitstreams which include downmixInstructions() payloads uses each permitted setting of *bsDownmixCoefficient* at least once. Base channel layouts containing LFE channels shall be chosen to ensure that downmix coefficients for LFE channels and non-LFE channels are exercised. Bitstreams designed to test *bsDownmixCoefficients* shall set *downmixCoefficientsPresent* to 1.

— The set of bitstreams which include downmixInstructionsV1() payloads shall use each permitted setting of *bsDownmixCoefficientV1* at least once and they shall use each permitted setting of *bsDownmixOffset* at least once. Bitstreams designed to test *bsDownmixCoefficients* shall set *downmixCoefficientsPresent* to 1.

— Some bitstreams have a *downmixCoefficientsPresent* value of zero.

— Some bitstreams have a *targetChannelCount* which is smaller than *baseChannelCount*.

— One or more bitstreams have a *targetChannelCount* which is identical to the *baseChannelCount.*

9.4.3.12 Ducking and fading test condition (Df)

9.4.3.12.1 General

This test condition shall be applied to verify the proper behaviour of the ducking and fading functionality of MPEG-D DRC and the proper signalling of album mode.

9.4.3.12.2 Test sequences

Bitstreams of this test condition shall be designed such that:

— Some bitstreams contain a DRC set for "Fade" (*drcSetEffect*=0x200). Test sequences shall be designed such that the fading DRC set is applied when the DRC tool is not in album mode (*albumMode*=0).

— Some bitstreams contain a DRC set for "Duck other" (*drcSetEffect*=0x400) and downmix metadata. Test sequences shall be designed such that the ducking DRC set is applied when the downmix is selected.

— Some bitstreams contain a DRC set for "Duck self" (*drcSetEffect*=0x800) and downmix metadata. Test sequences shall be designed such that the ducking DRC set is applied when the downmix is selected.

— Some bitstreams contain time-domain versions of all the bitstreams specified above and some bitstreams contain subband-domain versions of all the bitstreams specified above. Test sequences shall be designed such that both the time-domain and the subband-domain version are tested.

9.4.3.13 Peak limiter test condition (Pl)

9.4.3.13.1 General

This test condition shall be applied to verify the proper behaviour of the peak limiter tool.

9.4.3.13.2 Test sequences

Bitstreams of this test condition shall be designed such that:

— The audio output magnitude at the input of the peak limiter exceeds 6 dB FS.

— Some interface bitstreams set the value of *peakLimiterPresent* to one and others to zero.

— Different interface bitstreams set *outputPeakLevelMax* at least once to zero, a positive value, and a negative value.

— Some interface bitstreams do not set *outputPeakLevelMax* and set *peakLimiterPresent* at least once to one and zero.

9.4.3.14 Handling of extensions test condition (Ex)

9.4.3.14.1 General

This test condition shall be applied to ensure the proper handling of the extension payload mechanisms when parsing the bitstream in an MPEG-D DRC decoder.

An MPEG-D DRC decoder shall consume and discard extension payloads that it is not able to parse. Such payloads may be included in configuration payloads, interface payloads and loudness payloads. Table 100 lists such extension payloads.

Table 100 — Extension payloads

Extension payload
uniDrcGainExtension()
uniDrcConfigExtension()
loudnessInfoSetExtension()
uniDrcInterfaceExtension()

Several reserved types exist that may become valid types in the future. Table 101 lists such types. To ensure proper handling of the reserved types, an MPEG-D DRC decoder shall consume and discard payload bits associated with types that it cannot parse.

Table 101 — Reserved types

Type	Reserved values	Part of extension payload
uniDrcGainExtType	Larger than UNIDRCGAINEXT_TERM	uniDrcGainExtension()
uniDrcConfigExtType	Larger than UNIDRCCONFEXT_V1	uniDrcConfigExtension()
loudnessInfoSetExtType	Larger than UNIDRCLOUDEXT_EQ	loudnessInfoSetExtension()
parametricDrcType	Larger than PARAM_DRC_TYPE_LIM	parametricDrcInstructions()
uniDrcInterfaceExtType	Larger than UNIDRCINTERFACEEXT_EQ	uniDrcInterfaceExtension()

Reserved values exist for other fields as well. Any number of reserved values may become valid in the future. Conformant decoders shall be prepared to detect such reserved values and should take the proper action such as specified in Table 102.

Table 102 — Recommended decoder behaviour when a reserved value is present

Field	Part of payload	Decoder behaviour if reserved value is present
methodDefinition	loudnessInfo(), loudnessInfoV1()	Discard the loudnessInfo payload if it has a reserved methodDefinition value
measurementSystem	loudnessInfo(), loudnessInfoV1()	If value is larger than RMS_E, replace by default value
definedLayout	channelLayout()	Disable DRC tool if base channel count cannot be derived
speakerPosition	channelLayout()	Ignore reserved value
targetLayout	downmixInstructions(), downmixInstructionsV1()	Disable DRC tool if target channel count cannot be derived
bsDownmixOffset	downmixInstructionsV1()	Disable DRC tool if downmixInstructionsV1() with reserved bsDownmixOffset value are selected
drcCharacteristic	drcCoefficientsBasic(), drcCoefficientsUniDrc(), drcCoefficientsUniDrcV1()	Ignore DRCs that use a reserved drcCharacteristic
lfCornerFreqIndex, hfCornerFreqIndex	drcCoefficientsUniDrcV1()	Ignore dynamic EQs that use reserved values of xxCornerFreqIndex
lfFilterStrengthIndex, hfFilterStrengthIndex	drcCoefficientsUniDrcV1()	Ignore dynamic EQs that use reserved values of xxFilterStrengthIndex

Field	Part of payload	Decoder behaviour if reserved value is present
drcSetEffect	drcInstructionsBasic(), drcInstructionsUniDrc(), drcInstructionsUniDrcV1()	Ignore DRCs that have all zero bits in the non-reserved bit positions of drcSetEffect
frequencyRangeIndex	loudEqInstructions()	Ignore loudness EQs that use reserved values of frequencyRangeIndex
parametricDrcPresetId	parametricDrcInstructions()	Ignore parametric DRCs that use reserved values of parametricDrcPresetId
sideChainConfigType	drcCoefficientsParametricDrc()	Ignore parametric DRCs that use reserved values of sideChainConfigType
levelEstimKWeightingType	parametricDrcTypeFeedForward()	Ignore parametric DRCs that use reserved values of levelEstimKWeightingType
eqSetPurpose	eqInstructions()	Ignore EQs that have all zero bits in the non-reserved bit positions of eqSetPurpose
eqSubbandGainFormat	eqCoefficients()	Ignore EQs that use a reserved value for eqSubbandGainFormat

9.4.3.14.2 Test sequences

Bitstreams of this test condition shall be designed such that:

— A set of bitstreams includes all extension payloads given in Table 100 with non-zero size.

— A set of bitstreams uses a reserved type for each row of Table 101 with a corresponding extension payload of non-zero size and random content.

Annex A
(normative)

Tables

A.1 Coding of DRC gain values

Table A.1 — Coding of regular initial DRC gain values (*gainCodingProfile* == 0)

Encoding	Size	Mnemonic	gainInitial in [dB]	Range
$\{\sigma, \mu\}$	{1 bit, 8 bits}	{uimsbf, uimsbf}	$g_{DRC}(0) = (-1)^{\sigma}\,\mu 2^{-3}$	−31.875...31.875 dB, 0.125 dB step size

Table A.2 — Coding of initial DRC gain values for fading only (*gainCodingProfile* == 1)

Encoding	Size	Mnemonic	gainInitial in [dB]	Range
0	1 bit	bslbf	0	0
$\{1, \mu\}$	{1 bit, 10 bits}	{bslbf, uimsbf}	$g_{DRC}(0) = -(\mu + 1)2^{-3}$	−128...-0.125 dB, 0.125 dB step size

Table A.3 — Coding of initial DRC gain values for clipping prevention and ducking only (*gainCodingProfile* == 2)

Encoding	Size	Mnemonic	gainInitial in [dB]	Range
0	1 bit	bslbf	0	0
$\{1, \mu\}$	{1 bit, 8 bits}	{bslbf, uimsbf}	$g_{DRC}(0) = -(\mu + 1)2^{-3}$	−32...-0.125 dB, 0.125 dB step size

Table A.4 — Coding of regular DRC gain differences (*gainCodingProfile* $\in$ [0,1])

Codeword size [bits]	gainValueDelta binary encoding	gainDelta in [dB]
4	0x000	−2.0
9	0x039	−1.875
11	0x0E2	−1.750
11	0x0E3	−1.625
10	0x070	−1.500
10	0x1AC	−1.375
10	0x1AD	−1.250
9	0x0D5	−1.125
7	0x00F	−1.000
7	0x034	−0.875
7	0x036	−0.750

Codeword size [bits]	gainValueDelta binary encoding	gainDelta in [dB]
6	0x019	−0.625
5	0x002	−0.500
5	0x00F	−0.375
3	0x001	−0.250
2	0x003	−0.125
3	0x002	0.000
2	0x002	0.125
6	0x018	0.250
6	0x006	0.375
7	0x037	0.500
8	0x01D	0.625
9	0x0D7	0.750
9	0x0D4	0.875
5	0x00E	1.000

Table A.5 — Coding of DRC gain differences for clipping prevention and ducking only (*gainCodingProfile* == 2)

Codeword size [bits]	gainValueDelta binary encoding	gainDelta in [dB]
7	0x06A	−4.000
11	0x07A	−3.875
11	0x07B	−3.750
9	0x1AD	−3.625
10	0x03C	−3.500
9	0x1AC	−3.375
9	0x1A6	−3.250
9	0x0CD	−3.125
10	0x19E	−3.000
10	0x19F	−2.875
9	0x0CE	−2.750
9	0x1A7	−2.625
9	0x01F	−2.500
9	0x0CC	−2.375
8	0x0D2	−2.250
8	0x0AB	−2.125
8	0x0AA	−2.000
8	0x04F	−1.875

Codeword size [bits]	gainValueDelta binary encoding	gainDelta in [dB]
7	0x054	−1.750
7	0x068	−1.625
7	0x026	−1.500
7	0x006	−1.375
6	0x02B	−1.250
6	0x028	−1.125
6	0x002	−1.000
5	0x011	−0.875
5	0x00E	−0.750
4	0x00C	−0.625
4	0x009	−0.500
4	0x005	−0.375
4	0x003	−0.250
3	0x007	−0.125
4	0x001	0.000
4	0x00B	0.125
5	0x005	0.250
5	0x004	0.375
5	0x008	0.500
5	0x000	0.625
5	0x00D	0.750
5	0x00F	0.875
5	0x010	1.000
5	0x01B	1.125
6	0x012	1.250
6	0x018	1.375
6	0x029	1.500
7	0x032	1.625
8	0x04E	1.750
8	0x0D7	1.875
8	0x00E	2.000

A.2 Coding of time differences

Table A.6 — Coding of time differences with *nNodesMax* = N_{DRC} and *Z* = ceil($\log_2$(2nNodesMax*))**

Encoding	Size	Mnemonic	Time difference	Range
0x0	2 bits	bslbf	tDrcDelta = 1	1
{0x1,μ}	{2 bits, 2 bits}	{bslbf, uimsbf}	tDrcDelta = μ+2	2..5
{0x2,μ}	{2 bits, 3 bits}	{bslbf, uimsbf}	tDrcDelta = μ+6	6..13
{0x3,μ}	{2 bits, *Z* bits}	{bslbf, uimsbf}	tDrcDelta = μ+14	14..2*nNodesMax-1

A.3 Coding of slope steepness

Table A.7 — Coding of slope steepness (*gainInterpolationType* == 0)

Codeword size [bits]	Slope steepness binary encoding	Slope steepness
6	0x018	−3.0518
8	0x042	−1.2207
7	0x032	−0.4883
5	0x00A	−0.1953
5	0x009	−0.0781
5	0x00D	−0.0312
2	0x000	−0.0050
1	0x001	0.000
4	0x007	0.0050
5	0x00B	0.0312
6	0x011	0.0781
9	0x087	0.1953
9	0x086	0.4883
7	0x020	1.2207
7	0x033	3.0518

A.4 Coding of normalized crossover frequencies

Table A.8 — Coding of normalized crossover frequencies and associated filter coefficient parameters

crossoverFreqIndex	$f_{c,\mathrm{Norm}}$	γ	δ
0	2/1024	0.0000373252	0.9913600345
1	3/1024	0.0000836207	0.9870680830
2	4/1024	0.0001480220	0.9827947083
3	5/1024	0.0002302960	0.9785398263
4	6/1024	0.0003302134	0.9743033527
5	2/256	0.0005820761	0.9658852897
6	3/256	0.0012877837	0.9492662926
7	2/128	0.0022515827	0.9329321561
8	3/128	0.0049030350	0.9010958535
9	2/64	0.0084426929	0.8703307793
10	3/64	0.0178631928	0.8118317459
11	2/32	0.0299545822	0.7570763753
12	3/32	0.0604985076	0.6574551915
13	2/16	0.0976310729	0.5690355937
14	3/16	0.1866943331	0.4181633458
15	2/8	0.2928932188	0.2928932188

A.5 Coding of DRC gain extension types

Table A.9 — UniDrc gain extension types

Symbol	Value of uniDrcGainExtType	Purpose
UNIDRCGAINEXT_TERM	0x0	Termination tag
(reserved)	*(All remaining values)*	For future use

A.6 Coding of static DRC payload

A.6.1 Coding of top level fields of uniDrcConfig() and loudnessInfoSet()

Table A.10 — Coding of top level fields of uniDrcConfig() and loudnessInfoSet()

Field label	Encoding	Mnemonic	Decoded value	Description
bsSampleRate	μ 18 bits	uimsbf	$f_s = \mu + 1000$	Audio sample rate in [Hz]
downmixInstructionsCount, downmixInstructionsV1Count	μ 7 bits	uimsbf	$N_D = \mu$	Number of downmixInstructions() and downmixInstructionsV1() blocks, respectively
drcCoefficientsBasicCount, drcCoefficientsUniDrcCount, drcCoefficientsUniDrcV1Count	μ 3 bits	uimsbf	$N_C = \mu$	Number of drcCoefficients blocks
drcInstructionsBasicCount, drcInstructionsUniDrcCount, drcInstructionsUniDrcV1Count	μ 4, 6 bits	uimsbf	$N_I = \mu$	Number of drcInstructions blocks
loudnessInfoAlbumCount, loudnessInfoV1AlbumCount	μ 6 bits	uimsbf	$N_{LA} = \mu$	Number of loudnessInfo() and loudnessInfoV1() blocks for albums
loudnessInfoCount, loudnessInfoV1Count	μ 6 bits	uimsbf	$N_L = \mu$	Number of loudnessInfo() and loudnessInfoV1() blocks

A.6.2 Coding of loudnessInfoSet extension types

Table A.11 — loudnessInfoSet extension types

Symbol	Value of loudnessInfoSetExtType	Purpose
UNIDRCLOUDEXT_TERM	0×0	Termination tag
UNIDRCLOUDEXT_EQ	0×1	Extension for equalization
(reserved)	*(All remaining values)*	For future use
NOTE The extension type UNIDRCLOUDEXT_EQ is first available with the second edition of this document (ISO/IEC 23003-4:2019).		

A.6.3 Coding of DRC configuration extension types

Table A.12 — UniDrc configuration extension types

Symbol	Value of uniDrcConfigExtType	Purpose
UNIDRCCONFEXT_TERM	0×0	Termination tag
UNIDRCCONFEXT_PARAM_DRC	0×1	Parametric DRC
UNIDRCCONFEXT_V1	0×2	Efficient multi-band DRC coding, dynamic EQ, loudness EQ

Symbol	Value of uniDrcConfigExtType	Purpose
(reserved)	*(All remaining values)*	For future use
NOTE The extension types UNIDRCCONFEXT_PARAM_DRC and UNIDRCCONFEXT_V1 are first available with the second edition of this document (ISO/IEC 23003-4:2019).		

A.6.4 Coding of metadata that appears in multiple logical blocks of uniDrcConfig()

Table A.13 — Coding of metadata that appears in multiple logical blocks

Metadata field	Description
drcLocation	See 6.1.2.3.
drcSetId	A unique number for each drcInstructions block. 0x0 and 0×3F are reserved. A value of 0×3F in loudnessInfo() indicates that the loudness information can be applied to any DRC including no DRC.
dependsOnDrcSet	Same encoding as *drcSetId*.
downmixId	A unique number for each downmixInstructions() and downmixInstructionsV1() block. 0×00 and 0×7F are reserved. A value of 0×7F in drcInstructions indicates that it is permitted to apply the DRC set to the base layout or any downmix. In this case, the DRC set shall contain a single DRC gain sequence that is applied to all channels. If the *downmixId* is not present or if it has a value of zero, the DRC set is applied to the base layout. See also ISO/IEC 14496-12. If the V0 version of drcInstructionsUniDrc() includes a value of 0×7F and a downmix is active, the DRC set shall be applied to the downmix. For the V1 version, drcInstructionsUniDrcV1(), the *drcApplyToDownmix* flag indicates whether the DRC set shall be applied to the downmix or to the base layout.

A.6.5 Coded metadata in channelLayout()

Table A.14 — Coding of metadata in channelLayout()

Metadata field	Description
baseChannelCount	Number of channels in the base layout / original audio input (see 6.1.2.2).
definedLayout	See ChannelConfiguration in ISO/IEC 23091-3 (CICP). The channel order for each layout shall be as listed for the ChannelConfiguration in CICP.
speakerPosition	See OutputChannelPosition in ISO/IEC 23091-3 (CICP).

A.6.6 Coded metadata in downmixInstructions()

Table A.15 — Coding of metadata in downmixInstructions()

Metadata field	Description
targetLayout	See ChannelConfiguration in ISO/IEC 23091-3 (CICP).
bsDownmixOffset	See ISO/IEC 14496-12 (ISO base media file format).
bsDownmixCoefficient	See ISO/IEC 14496-12 (ISO base media file format).
bsDownmixCoefficientV1	See ISO/IEC 14496-12 (ISO base media file format).

A.6.7 Coded metadata in drcCoefficientsBasic() and drcCoefficientsUniDrc()

Table A.16 — Coding of metadata in drcCoefficientsBasic(), drcCoefficientsUniDrc(), and drcCoefficientsUniDrcV1()

Metadata field	Description
drcLocation	See Table 2.
drcFrameSizePresent	A value of 1 indicates that *bsDrcFrameSize* value is present.
bsDrcFrameSize	Encoding of DRC frame size according to Table A.17.
drcCharacteristicLeftPresent, drcCharacteristicRightPresent	Flag to indicate if a characteristic is defined (1) or not (0).
characteristicLeftCount, characteristicRightCount	Number of defined characteristics.
characteristicFormat	Format of characteristic: sigmoidal (0), segment-wise (1) representation.
bsGainLeft, bsGainRight	Encoded DRC gain according to Table A.24 and Table A.25.
bsIoRatioLeft, bsIoRatioRight	Encoded I/O ratio for sigmoidal characteristic according to Table A.26.
bsExpLeft, bsExpRight	Encoded exponent for sigmoidal characteristic according to Table A.27.
flipSignLeft, flipSignRight	Flag to indicate if the DRC characteristic is multiplied by -1 (1) or not (0).
bsCharNodeCount	Encoded node count for segment-wise representation according to Table A.28.
bsNodeLevelDelta	Encoded differential DRC gain for this node according to Table A.29.
bsNodeGain	Encoded DRC gain for this node according to Table A.30.
shapeFiltersPresent	A value of 1 indicates that shape filter parameters are present.
shapeFilterCount	Number of shape filter parameter sets.
lfCutFilterPresent, lfBoostFilterPresent, hfCutFilterPresent, hfBoostFilterPresent,	A value of 1 indicates that the corresponding shape filter parameters are present.
lfCornerFreqIndex, hfCornerFreqIndex	Index of corner frequency parameter for low-frequency and high-frequency shape filters, respectively.
lfFilterStrengthIndex, hfFilterStrengthIndex,	Index of filter strength parameter for low-frequency and high-frequency shape filters, respectively.
gainSequenceCount	Number of gain sequences in the uniDrcGain() payload.
gainSetCount	Number of gain sequence sets. A set for multi-band DRC can include multiple gain sequences.
gainCodingProfile	See Table A.18.
gainInterpolationType	See Table A.19.
fullFrame	A value of 1 signals that the last node is always at the end of the frame. In low-delay mode, it shall be 1.
timeAlignment	A bit field that indicates whether the gain sample is aligned with the end (0) or the center (1) of the *deltaTmin* interval.
delayMode	A bit field that indicates whether the received gains are applied immediately or with a delay of one frame (see Table A.20).
timeDeltaMinPresent	A value of 1 indicates that a *bsTimeDeltaMin* value is present.

Metadata field	Description
bsTimeDeltaMin	This field indicates the custom time resolution of a DRC sequence which overrides the default *deltaTmin* value. The values are encoded according to Table A.21.
bandCount	The number of DRC bands for this gain set.
drcBandType	A bit field, which signals if the "crossoverFreqIndex-syntax" (1) or the "startSubBandIndex-syntax" (0) should be used. If multi-band DRC gains are applied in the time-domain by using the multi-band DRC filter bank like specified in 6.4.12, only the "crossoverFreqIndex-syntax" is allowed. If the "startSubBandIndex-syntax" is used, the frequency smoothing/fading with overlap weights according to Table 30 is not applied.
indexPresent	A value of 1 indicates that a *bsIndex* value is present.
bsIndex	Encoded gain sequence index that is copied into *gainSequenceIndex* which refers to the gain sequences in uniDrcGain() numbered in the order of appearance.
drcCharacteristicPresent	Flag indicating whether a source characteristic for the gain sequence is present (1) or not (0).
drcCharacteristicFormatIsCICP	Flag indicating whether the index of the characteristic is based on ISO/IEC 23008-1, (1) or not (0).
drcCharacteristic	A value of 0 means that the DRC characteristic is undefined for a DRC sequence. Values > 11 are reserved. See ISO/IEC 23091-3 (CICP).
drcCharacteristicLeftIndex, drcCharacteristicRightIndex	Index of source DRC characteristic of the gain sequence referring to *characteristicLeftIndex* and *characteristicRightIndex* in drcCoefficientsUniDrcV1().
crossoverFreqIndex	See Table A.8.
startSubBandIndex	Zero-based sub-band index for an available sub-band domain. The field is used to signal the start index for a specific DRC band.

Table A.17 — Coding of *bsDrcFrameSize* field

Field	Encoding	Mnemonic	drcFrameSize	Range
bsDrcFrameSize	μ 15 bits	uimsbf	$M_{DRC} = \mu+1$	DRC frame size in units of audio sample intervals. Range: $1...2^{15}$

Table A.18 — Coding of *gainCodingProfile* field

Value	Meaning
0	Regular gain coding
1	Fading gain coding
2	Clipping prevention and ducking gain coding
3	Constant gain (no gain sequence is transmitted)

Table A.19 — Coding of *gainInterpolationType* field

Value	Meaning
0	Spline interpolation
1	Linear interpolation

Table A.20 — Coding of *delayMode* field

Value	Meaning
0	Regular delay (Gains are applied with a delay of one frame)
1	Low delay (Received gains are applied immediately)

Table A.21 — Coding of *bsTimeDeltaMin* field

Field	Encoding	Mnemonic	timeDeltaMin	Range
bsTimeDeltaMin	μ 11 bits	uimsbf	deltaTmin = μ+1	*deltaTmin* in units of audio sample intervals. Range: 1…2^{11}

Table A.22 — Coding of *drcCharacteristic* field

See ISO/IEC 23091-3

Table A.23 — Coding of *startSubBandIndex* field

Field	Encoding	Mnemonic	Decoded value	Range
startSubBandIndex	μ 10 bits	uimsbf	startSubBandIndex = μ	*startSubBandIndex* in zero-based indices of the sub-band domain. Range: 0…2^{10}-1

Table A.24 — Coding of *bsGainRight*

Encoding	Size	Mnemonic	gainDbRight [dB]	Range
μ	6 bits	uimsbf	$-\mu$	-63…0 dB, 1 dB step size

Table A.25 — Coding of *bsGainLeft*

Encoding	Size	Mnemonic	gainDbLeft [dB]	Range
μ	6 bits	uimsbf	μ	0…63 dB, 1 dB step size

Table A.26 — Coding of *bsIoRatioLeft* and *bsIoRatioRight*

Encoding	Size	Mnemonic	ioRatioLeft, ioRatioRight	Range
μ	4 bits	uimsbf	$0.05 + 0.15\mu$	0.05…2.3, step size of 0.15

Table A.27 — Coding of *bsExpLeft* and *bsExpRight*

Encoding	Size	Mnemonic	expLeft, expRight	Range
μ	4 bits	uimsbf	$1 + 2\mu$; $\mu < 15$ ∞ ; else	1…29, step size of 2 and ∞

Table A.28 — Coding of *bsCharNodeCount*

Encoding	Size	Mnemonic	characteristicNodeCount	Range
μ	2 bits	uimsbf	$\mu + 1$	1…4

Table A.29 — Coding of *bsNodeLevelDelta*

Encoding	Size	Mnemonic	nodeLevelDelta [dB]	Range
μ	5 bits	uimsbf	$\mu + 1$	1…32 dB, 1 dB step size

Table A.30 — Coding of *bsNodeGain*

Encoding	Size	Mnemonic	nodeGain [dB]	Range
μ	8 bits	uimsbf	$0.5\mu - 64$	-64…63.5 dB, 0.5 dB step size

Table A.31 — Coding of coefficient boundary $y_{1,bound}$ for LF shaping filters depending on *lfCornerFreqIndex* and *lfFilterStrengthIndex* in drcCoefficientsUniDrcV1 ()

lfCornerFreqIndex	lfFilterStrengthIndex			
	0	1	2	3 (reserved)
0	-0.994	-0.996	-1.000	*reserved*
1	-0.990	-0.995	-0.999	*reserved*
2	-0.980	-0.989	-0.996	*reserved*
3	-0.970	-0.983	-0.994	*reserved*
Remaining indexes reserved	*reserved*	*reserved*	*reserved*	*reserved*

Table A.32 — Coding of coefficient boundary $y_{1,bound}$ for HF shaping filters depending on *hfCornerFreqIndex* and *hfFilterStrengthIndex* in drcCoefficientsUniDrcV1 ()

hfCornerFreqIndex	hfFilterStrengthIndex			
	0	1	2	3 (reserved)
0	0.150	0.750	1.050	*reserved*
1	0.430	0.870	1.070	*reserved*
2	0.600	0.920	1.070	*reserved*
3	0.800	1.000	1.060	*reserved*
4	0.900	1.040	1.073	*reserved*
Remaining indexes reserved	*reserved*	*reserved*	*Reserved*	*reserved*

Table A.33 — Coding of coefficient gain offset g_{offset} for LF shaping filters depending on *lfCornerFreqIndex* and *lfFilterStrengthIndex* in drcCoefficientsUniDrcV1 ()

lfCornerFreqIndex	lfFilterStrengthIndex			
	0	1	2	3 (reserved)
0	3.0	2.0	1.2	*reserved*
1	3.0	2.0	1.5	*reserved*
2	3.0	2.0	2.0	*reserved*
3	3.0	2.0	2.0	*reserved*
Remaining indexes reserved	*reserved*	*reserved*	*reserved*	*reserved*

Table A.34 — Coding of coefficient gain offset g_{offset} for HF shaping filters depending on *hfCornerFreqIndex* and *hfFilterStrengthIndex* in drcCoefficientsUniDrcV1 ()

hfCornerFreqIndex	hfFilterStrengthIndex			
	0	1	2	3 (reserved)
0	4.50	6.00	3.50	*reserved*
1	3.70	4.00	2.70	*reserved*
2	3.00	3.50	2.00	*reserved*
3	2.00	2.50	1.50	*reserved*
4	1.50	2.00	1.31	*reserved*
Remaining indexes reserved	*Reserved*	*reserved*	*reserved*	*reserved*

Table A.35 — Coding of Radius *r* for LF shaping filters depending on *lfCornerFreqIndex* in drcCoefficientsUniDrcV1 ()

lfCornerFreqIndex	*r*
0	0.988
1	0.980
2	0.960
3	0.940
Remaining indexes reserved	*reserved*

Table A.36 — Coding of normalized cutoff frequency $f_{c,norm}$ and radius *r* for HF shaping filters depending on *hfCornerFreqIndex* in drcCoefficientsUniDrcV1 ()

hfCornerFreqIndex	$f_{c,norm}$	*r*
0	0.15	0.45
1	0.20	0.40
2	0.25	0.35
3	0.35	0.30
4	0.45	0.30
Remaining indexes reserved	*reserved*	*reserved*

A.6.8 Coded metadata in drcInstructionsBasic() and drcInstructionsUniDrc()

Table A.37 — Coding of metadata in drcInstructionsBasic(),drcInstructionsUniDrc(), and drcInstructionsUniDrcV1()

Metadata field	Description
drcSetId	Index to identify this DRC set. The value of 0 and 0×3F is reserved.
drcSetComplexityLevel	Value that indicates the computational complexity level of the associated DRC set per audio channel. The complexity level is used to determine if a decoder is capable of applying the DRC set (see 6.9).
drcLocation	Location of DRC gain sequence (see Table 2).
downmixIdPresent	A flag that has a value of 1 if *downmixId* is present.
downmixId	Index to identify a downmix that is associated with this DRC set.
additionalDownmixIdPresent	A flag that has a value of 1 if additional downmix IDs are present.
additionalDownmixIdCount	Number of additional downmix IDs connected with one DRC set. Only one DRC channel group is allowed if an additional downmix ID is present and the DRC is applied to a downmix.

Metadata field	Description
additionalDownmixId	Additional downmix ID connected with a *drcSetId*. This field may be used to declare additional target layouts for which one single DRC set provides suitable gains, e.g. for clipping prevention of multiple target layouts with the same DRC set. Only one DRC channel group is allowed if an additional downmix ID is present and the DRC is applied to a downmix.
drcApplyToDownmix	A flag that has a value of 1 to indicate that the DRC set is applied to the downmix. Otherwise it is applied to the base layout. Since this flag is only present in drcIntructionsUniDrcV1() payloads, for the other payloads the DRC is applied to the downmix if a *downmixId* is present that is different from 0×0.
drcSetEffect	Declares the effect of the DRC according to Table A.45.
limiterPeakTargetPresent	A flag that has a value of 1 if a *bsLimiterPeakTarget* value is present.
bsLimiterPeakTarget	Encoded limiter peak target value according to Table A.40.
drcSetTargetLoudnessPresent	A flag that has a value of 1 if a *bsDrcSetTargetLoudnessValueUpper* value is present.
bsDrcSetTargetLoudnessValueUpper	A field which contains the upper limit of the target loudness of a DRC set. The default value is 0 dB. The values are encoded according to Table A.44.
bsDrcSetTargetLoudnessValueLowerPresent	A flag that has a value of 1 if a *bsDrcSetTargetLoudnessValueLower* value is present.
bsDrcSetTargetLoudnessValueLower	A field which contains the lower limit of the target loudness of a DRC set. The default value is -63 dB. The values are encoded according to Table A.44.
dependsOnDrcSetPresent	A flag that has a value of 1 if a *dependsOnDrcSet* value is present.
dependsOnDrcSet	Indicates the drcSetId of the DRC set this DRC depends on. Same encoding as *drcSetId*.
noIndependentUse	A flag which signals that the DRC set can only be used in combination with another DRC set as indicated by the *dependsOnDrcSet* field.
requiresEq	A flag which signals that the DRC set can only be used in combination with an EQ set, i.e. if *requiresEq*==1 it is not permitted to use this DRC set without an associated EQ set, and a DRC tool that does not support EQ shall ignore this DRC set. For combined DRC sets with a primary and dependent set, the *requiresEq* field of the dependent set is ignored.
duckingScalingPresent	A flag that has a value of 1 if a *bsDuckingScaling* value is present.
bsDuckingScaling	Encoded scaling value for ducking.
repeatParameters	A flag that has a value of 1 if a *bsRepeatParametersCount* value is present.
bsRepeatParametersCount	Encoded count value indicating how many times the same *bsGainSetIndex* shall be repeatedly used for subsequent channels (see Table A.39).

Metadata field	Description
bsGainSetIndex	A unique number which specifies which DRC gain set should be assigned to which channel/object of the configuration specified in *downmixId*. The reserved value 0×00 indicates that the assigned channel will not be processed by the DRC tool. All other values indicate the assigned DRC gain set index according to *gainSetIndex*. Due to the reserved value, the number of DRC gain sets in one location cannot be more than 63.
repeatGainSetIndex	Flag to indicate whether the gain set index is repeated (1) or not (0).
bsRepeatGainSetIndexCount	A field which declares that the current gain set index should be repeated for multiple channels. The assignment for-loop continues at position "i = i + *repeatGainSetCount*". *bsRepeatGainSetIndexCount* is encoded according to Table A.39. If a sequence index should be repeated more than 32 times, *bsGainSetIndex* has to be signaled again.
targetCharacteristicLeftPresent, targetCharacteristicRightPresent	Flag to indicate whether a target DRC characteristic is defined (1) or not (0).
targetCharacteristicLeftIndex , targetCharacteristicRightIndex	Index of left and right target DRC characteristic, respectively. The index refers to *characteristicLeftIndex* and *characteristicRightIndex*, in drcCoefficientsUniDrcV1(), respectively. The index value of 0 is reserved.
gainScalingPresent	A flag that has a value of 1 if a scaling values are present.
bsAttenuationScaling, bsAmplificationScaling	Encoded scaling values according to Table A.42.
gainOffsetPresent	A flag that has a value of 1 if an offset value is present.
bsGainOffset	Encoded offset value according to Table A.43.
shapeFilterPresent	A value of 1 indicates that a shape filter index is present.
shapeFilterIndex	1-based index that refers to a specific shape filter parameter set in the drcCoefficientsUniDrcV1() payload as indicated by the *shapeFIlterIndex* value in Table 68.

Table A.38 — Coding of *bsGainSetIndex*

Encoding	Size	Mnemonic	gainSetIndex	Range
μ	6 bits	uimsbf	$\mu-1$ if $\mu > 0$, else undefined	0…62

Table A.39 — Coding of *bsRepeatParametersCount* and *bsRepeatGainSetIndexCount* field

Encoding	Size	Mnemonic	repeatParametersCount, repeatGainSetIndexCount	Range
μ	5 bits	uimsbf	$N_R = \mu+1$	1…32

Table A.40 — Coding of *bsLimiterPeakTarget* field

Encoding	Size	Mnemonic	limiterPeakTarget in [dBFS]	Range
μ	8 bits	uimsbf	$L_{PT} = -\mu 2^{-3}$	−31.875 ... 0 dBFS, 0.125 dB step size

Table A.41 — Coding of *bsDuckingScaling* field

Encoding	Size	Mnemonic	duckingScaling	Range
{σ,μ}	{1 bit, 3 bits}	{uimsbf, uimsbf}	$w = 1 + (-1)^{\sigma}(1+\mu)2^{-3}$	1 ± [0.125...1], 0.125 step size

Table A.42 — Coding of *bsAttenuationScaling* and *bsAmplificationScaling* field

Encoding	Size	Mnemonic	attenuationScaling, amplificationScaling	Range
μ	4 bits	uimsbf	$w = \mu 2^{-3}$	0... 1.875, 0.125 step size

Table A.43 — Coding of *bsGainOffset* field

Encoding	Size	Mnemonic	gainOffset in [dB]	Range
{σ,μ}	{1 bit, 5 bits}	{uimsbf, uimsbf}	$g_{\text{offs}} = (-1)^{\sigma}(1+\mu)2^{-2}$	±[0.25...8], 0.25 step size

Table A.44 — Coding of *bsDrcSetTargetLoudnessValueUpper/-Lower* field

Encoding	Size	Mnemonic	drcSetTargetLoudnessValue in [LKFS]	Range
μ	6 bits	uimsbf	drcSetTargetLoudnessValue = μ − 63	−63 ... 0 LKFS, 1 dB step size

Table A.45 — Coding of *drcSetEffect* field. A bit value of 1 indicates that the effect is present

bit position	drcSetEffect	Short name	Description
1 (LSB)	Late night	“Night”	For quiet environment, listening at low level, avoiding to disturb others.
2	Noisy environment	“Noisy”	Optimized to get the best experience in noisy environments, for instance by amplifying soft sections.
3	Limited playback range	“Limited”	Reduced dynamic range to improve quality on playback devices with limited dynamic range.
4	Low playback level	“LowLevel”	Listening at a low playback level.
5	Dialog enhancement	“Dialog”	The main effect is a more prominent dialogue within the content.

bit position	drcSetEffect	Short name	Description
6	General compression	"General"	A DRC effect that reduces the dynamic range and is applicable to multiple playback scenarios.
7	Dynamic expansion	"Expand"	Dynamics enhancement.
8	Artistic effect	"Artistic"	To create an artistic sound effect.
9	Clipping prevention	"Clipping"	The main purpose is clipping prevention.
10	Fade-in/fade-out	"Fade"	Fade-in and fade-out envelope for gapless content (applicable when not playing in Album mode). It has no dynamic compression.
11	Ducking other	"Duck other"	An effect that attenuates all audio content except for the channelGroup it is associated with. It has no dynamic compression.
12	Ducking self	"Duck self"	An effect that attenuates all channelGroups that it is associated with. It is identical to "Ducking other", however, it has the same channelGroup assignment as regular DRC gains.
Remaining values are reserved			
NOTE Any information on DRC sets with "Ducking" effect in this document shall refer to both "Ducking other" and "Ducking self" if not stated otherwise.			

A.6.9 Coded metadata in loudnessInfo()

Table A.46 — Coding of *bsSamplePeakLevel* field

Encoding	Size	Mnemonic	samplePeakLevel in [dBFS]	Approximate range
μ	12 bits	uimsbf	$L_{sp} = \begin{cases} \text{"undefined"; if } \mu == 0 \\ L_{sp} = 20 - \mu 2^{-5}\text{; else} \end{cases}$	−107 ... 20, 0.0312 step size

Table A.47 — Coding of *bsTruePeakLevel* field (True Peak[4])

Encoding	Size	Mnemonic	truePeakLevel in [dBTP]	Approximate range
μ	12 bits	uimsbf	$L_{sp} = \begin{cases} \text{"undefined"; if } \mu == 0 \\ L_{sp} = 20 - \mu 2^{-5}\text{; else} \end{cases}$	−107 ... 20, 0.0312 step size

Table A.48 — Coding of *methodValue* field

Encoding	Format	methodDefinition	methodValue	Approx. range
μ	8 uimsbf	0, ..., 5	L = -57.75 + $\mu 2^{-2}$	−57.75 ... 6, 0.25 step size
See Table A.52	8 uimsbf	6	See Table A.52	0 ... 121

Encoding	Format	methodDefinition	methodValue	Approx. range
μ	5 uimsbf	7	$L = 80 + \mu$	80 ... 111 dB
μ	2 uimsbf	8	0x0: "not indicated" 0x1: "large room, X curve monitor"[1] 0x2: "small room, flat monitor"[1] 0x3: "reserved"	n/a
μ	8 uimsbf	9	$L = -116 + \mu 2^{-1}$	−116 ... 11.5 LKFS, 0.5 step size

Table A.49 — Coding of *methodDefinition* field in *loudnessInfo*()

Value	Value type	Meaning
0	n/a	unknown/other
1	Loudness	program loudness (programLoudness as defined in ISO/IEC 23091-3)
2	Loudness	anchor loudness (anchorLoudness as defined in ISO/IEC 23091-3)
3	Loudness	maximum of the range, i.e. the 95th percentile of the loudness distribution according to EBU R-128[8],[7]
4	Loudness	maximum momentary loudness, measured using a 0.4s window according to ITU-R BS.1771–1[5] or EBU R-128[8],[6]
5	Loudness	maximum short-term loudness, measured using a 3s window according to ITU-R BS.1771–1[5] or EBU R-128[8],[6]
6	Loudness range	loudness range derived from EBU R-128[8],[7]
7	Sound pressure level	production mixing level measured according to[1]
8	Index	production room type according to[1]
9	Loudness	short-term loudness, measured using a 3s window according to ITU-R BS.1771–1[5] or EBU R-128[8],[6] The 3s window shall include the current DRC frame.
Remaining values are reserved		

Table A.50 — Coding of *measurementSystem* field in loudnessInfo()

Value	Meaning
0	Unknown/other
1	EBU R-128[8]
2	ITU-R BS.1770–4[4]
3	ITU-R BS.1770–4 with pre-processing. The pre-processor is a 4th order Linkwitz-Riley filter with a cutoff frequency of 500 Hz.
4	User
5	Expert/panel
6	ITU-R BS.1771–1[5]
7 *(reserved, not permitted)*	Reserved Measurement System A (RMS_A)

Value	Meaning
8 *(reserved, not permitted)*	Reserved Measurement System B (RMS_B)
9 *(reserved, not permitted)*	Reserved Measurement System C (RMS_C)
10 *(reserved, not permitted)*	Reserved Measurement System D (RMS_D)
11 *(reserved, not permitted)*	Reserved Measurement System E (RMS_E)
Remaining values are reserved	

Table A.51 — Coding of *reliability* field in loudnessInfo()

Value	Meaning
0	Reliability is unknown
1	Value is reported/imported but unverified
2	Value is a "not to exceed" ceiling
3	Value is measured and accurate

Table A.52 — Coding algorithm for *loudnessRange* in dB

```
methodValue(loudnessRange) {
    if (loudnessRange < 0.0f)
        return 0;
    else if(loudnessRange <= 32.0f)
        return (UInt8)(4.0f*loudnessRange + 0.5f);
    else if(loudnessRange <= 70.0f)
        return (UInt8)(2.0f*(loudnessRange - 32.0f) + 0.5f) + 128;
    else if(loudnessRange < 121.0f)
        return (UInt8)((loudnessRange - 70.0f) + 0.5f) + 204;
    else
        return 255;
}
```


A.6.10 Coded metadata in drcCoefficientsParametricDrc()

Table A.53 — Coding of metadata in drcCoefficientsParametricDrc()

Metadata field	Description
drcLocation	See 6.1.2.3. If a drcCoefficientsUniDrc() block for the same *drcLocation* is present, the first *gainSetIndex* referring to drcCoefficientsParametricDrc() starts with an offset of the *gainSetCount* defined in drcCoefficientsUniDrc(). The same holds for a drcCoefficientsUniDrcV1() block if present.
parametricDrcFrameSizeFormat	A field that signals whether parametricDrcFrameSize is coded by *bsParametricDrcFrameSize* (0) or by *bsDrcFrameSize* (1).
bsParametricDrcFrameSize	A field that signals the processing frame size of parametric DRCs. The values are encoded according to Table A.54.
bsDrcFrameSize	A field that signals the processing frame size of parametric DRCs. The values are encoded according to Table A.17.

Metadata field	Description
bsParametricDrcDelayMax	A field that specifies a delay value greater than or equal to the maximum delay of any applicable parametric DRC set combination in the stream. The values are encoded according to Table A.55.
resetParametricDrc	The default value is 0. A value of 1 indicates that the parametric DRC shall be reset (e.g. for streaming scenarios). See also 6.6.
parametricDrcGainSetCount	Number of parametric DRC gain sets.
parametricDrcId	A unique 0-based identifier that refers to a specific parametricDrcInstructions() block.
sideChainConfigType	A field that defines the side-chain configuration of a parametric DRC gain set. The values are encoded according to Table A.56. For a value of 1, the channel weights can be customized based on an applicable *downmixId*. For all other values, the side-chain input signal is defined by the *drcChannelGroup* the DRC gains set is assigned to within a drcInstructionsUniDrc() or drcInstructionsUniDrcV1() block. If a value is not applicable for a parametric DRC instance, sideChainConfigType shall be set to 0, which is the default. For drcInstructionsUniDrcV1(), the applicable *downmixId* is 0 (baseLayout) for *drcApplyToDownmix*=0.
downmixId	See Table A.13.
levelEstimChannelWeightFormat	A field that signals if a channel map (0) or a channel weight map (1) is present for the definition of the side-chain signal of a parametric DRC instance.
addChannel	A value of 1 indicates that the current channel shall be added to the side-chain signal of a parametric DRC instance.
bsChannelWeight	A field that defines a weighting factor for the current channel for addition to the side-chain signal of a parametric DRC instance. The values are encoded according to Table A.57.
bsDrcInputLoudness	A field that signals the input loudness for a parametric DRC gain set. The values are encoded according to Table A.58. The input loudness is required for normalization of the DRC side-chain input level.

Table A.54 — Coding of *bsParametricDrcFrameSize* field

Field	Encoding	Mnemonic	parametricDrcFrameSize in [samples]	Range
bsParametricDrcFrameSize	μ 4 bits	uimsbf	$M_{parametricDRC} = 2^{\mu}$	1…2^{15} audio sample intervals, variable step size.

Table A.55 — Coding of *bsParametricDrcDelayMax* field

Field	Encoding	Mnemonic	parametricDrcDelayMax in intervals of audio sample rate	Range
bsParametricDrcDelayMax	{μ, ν} {5 bits, 3 bits}	{uimsbf, uimsbf}	$16\mu 2^{\nu}$	0...63488, variable step size.

Table A.56 — Coding of *sideChainConfigType* field

Value	Meaning
0 (default)	Side-chain signal defined by all channels of the processed *drcChannelGroup* and *channelWeight*[] set to 0 dB.
1	Custom configuration of side-chain signal based on an applicable *downmixId*.
2	Side-chain signal defined by all channels of the processed *drcChannelGroup* and *channelWeight*[] set according to ITU-R BS.1770-4.
Remaining values are reserved.	

Table A.57 — Coding of *bsChannelWeight* field

Field	Encoding	Mnemonic	μ	channelWeight [dB]	Range
bsChannelWeight	μ 4 bits	uimsbf	0	10.0	- ∞ ... 10.0 dB, variable step size.
			1	6.0	
			2	4.5	
			3	3.0	
			4	1.5	
			5	0.0	
			6	-1.5	
			7	-3.0	
			8	-4.5	
			9	-6.0	
			10	-10.0	
			11	-15.0	
			12	-20.0	
			13	-30.0	
			14	-40.0	
			15	-∞	

Table A.58 — Coding of *bsDrcInputLoudness* field

Field	Encoding	Mnemonic	drcInputLoudness in [dB]	Range
bsDrcInputLoudness	μ 8 bits	uimsbf	$L_{reference} = -57.75 + \mu 2^{-2}$	-57.75 ... 6 dB, 0.25 dB step size.

A.6.11 Coded metadata in parametricDrcInstructions()

Table A.59 — Coding of metadata in parametricDrcInstructions()

Metadata field	Description
parametricDrcId	A unique 0-based identifier for each parametricDrcInstructions() block.
bsParametricDrcLookAhead	A field that signals the audio delay that should be applied for a parametric DRC instance. See Table A.60. If not present, a default value is set based on *parametricDrcType*.
parametricDrcPresetIdPresent	A field that indicates whether the parametric DRC type is defined by *parametricDrcPresetId* (1) or not (0).
parametricDrcPresetId	A unique 0-based identifier that is mapped to a *parametricDrcType* and predefined settings according to Table A.62. If an unsupported value is received, the corresponding parametric DRC instance shall be disabled.
parametricDrcType	A field that signals the parametric DRC type. See Table A.61.

Table A.60 — Coding of *bsParametricDrcLookAhead* field

Field	Encoding	Mnemonic	parametricDrcLookAhead in [ms]	Range
bsParametricDrcLookAhead	μ 7 bits	uimsbf	$T_{lookAhead} = \mu$	0...127 ms, 1 ms step size.

Table A.61 — Coding of *parametricDrcType*

Symbol	Value of parametricDrcType	Purpose
PARAM_DRC_TYPE_FF	0×0	Parameters defined by parametricDrcTypeFeedForward() block (see 6.6.2.3 and 6.6.3.1).
PARAM_DRC_TYPE_LIM	0×1	Parameters defined by parametricDrcTypeLimiter() block (see 6.6.2.4 and 6.6.3.2).
(reserved)	*2-6*	For ISO use.
(reserved)	*7*	For use outside of ISO scope.

Table A.62 — Mapping of *parametricDrcPresetId* to *parametricDrcType*

parametricDrcPresetId	parametricDrcType	Parameters	Description
0	PARAM_DRC_TYPE_FF	drcCharacteristic = 7	DRC curve definition according to ISO/IEC 23091-3. DRC gain smoothing time constants according to Table A.77.
1	PARAM_DRC_TYPE_FF	drcCharacteristic = 8	
2	PARAM_DRC_TYPE_FF	drcCharacteristic = 9	
3	PARAM_DRC_TYPE_FF	drcCharacteristic = 10	
4	PARAM_DRC_TYPE_FF	drcCharacteristic = 11	
5	PARAM_DRC_TYPE_LIM	—	Default parameters as specified in 6.6.3.2.
6-127	*(reserved)*	*(reserved)*	*(reserved)*

A.6.12 Coded metadata in parametricDrcTypeFeedForward()

Table A.63 — Coding of metadata in parametricDrcTypeFeedForward()

Metadata field	Description
levelEstimKWeightingType	A field that indicates the state of the pre-filter and RLB filter for level estimation according to ITU-R BS.1770-4. If not present, the default value is 2. See Table A.64.
bsLevelEstimIntegrationTime	Level estimation integration time. If not present, the default value is *parametricDrcFrameSize*. See Table A.65.
drcCurveDefinitionType	A field that indicates whether the DRC curve is defined by a *drcCharacteristic* index (0) or by a DRC curve parametrization.
drcCharacteristic	*drcCharacteristic* index and DRC curve definition according to ISO/IEC 23091-3. Following *drcCharacteristic* indices are currently supported for the usage with parametricDrcType == PARAM_DRC_TYPE_FF: {7,8,9,10,11}. If an unsupported value is received, the corresponding parametric DRC instance shall be disabled.
bsNodeCount	Number of curve nodes. See Table A.66.
bsNodeLevelInitial	Input level of first node in dB. See Table A.67.
bsNodeLevelDelta	Input level distance to next node in dB. See Table A.68.
bsNodeGain	Node output gain in dB. See Table A.69.
drcGainSmoothParametersPresent	A field that indicates whether custom smoothing parameters are provided (1) or whether smoothing parameters matching to a present *drcCharacteristic* index shall be used (0) (see Table A.77). If *drcCharacteristic* is not present, the default smoothing parameters shall be used.
bsGainSmoothAttackTimeSlow	Slow attack time for DRC gain smoothing in ms. See Table A.70.
bsGainSmoothReleaseTimeSlow	Slow release time for DRC gain smoothing in ms. See Table A.71.
bsGainSmoothAttackTimeFast	Fast attack time for DRC gain smoothing in ms. See Table A.72.
bsGainSmoothReleaseTimeFast	Fast release time for DRC gain smoothing in ms. See Table A.73.

Metadata field	Description
bsGainSmoothAttackThreshold	Fast attack threshold for DRC gain smoothing in dB. See Table A.74.
bsGainSmoothReleaseThreshold	Fast release threshold for DRC gain smoothing in dB. See Table A.75.
bsGainSmoothHoldOff	Hold counter for DRC gain smoothing. See Table A.76.

Table A.64 — Coding of *levelEstimKWeightingType* field

Value	Meaning
0	Pre-filter OFF, RLB filter OFF
1	Pre-filter OFF, RLB filter ON
2	Pre-filter ON, RLB filter ON
3	Reserved

Table A.65 — Coding of *bsLevelEstimIntegrationTime* field

Field	Encoding	Mnemonic	levelEstimIntegrationTime in [samples]	Range
bsLevelEstimIntegrationTime	μ 6 bits	uimsbf	$T_{integration} = (\mu+1)M_{parametricDRC}$	1 ... 64 times $M_{parametricDRC}$ audio sample intervals.

Table A.66 — Coding of *bsNodeCount* field

Field	Encoding	Mnemonic	nodeCount	Range
bsNodeCount	μ 3 bits	uimsbf	$N_{nodes} = \mu + 2$	2 ... 9.

Table A.67 — Coding of *bsNodeLevelInitial* field

Field	Encoding	Mnemonic	nodeLevelInitial in [dB]	Range
bsNodeLevelInitial	μ 6 bits	uimsbf	$L_{nodes,initial} = -11 - \mu$	-74...-11 dB, 1 dB step size.

Table A.68 — Coding of *bsNodeLevelDelta* field

Field	Encoding	Mnemonic	nodeLevelDelta in [dB]	Range
bsNodeLevelDelta	μ 5 bits	uimsbf	$\Delta L_{node} = 1 + \mu$	1...32 dB, 1 dB step size.

Table A.69 — Coding of *bsNodeGain* field

Field	Encoding	Mnemonic	nodeGain in [dB]	Range
bsNodeGain	μ 6 bits	uimsbf	$G_{node} = \mu - 39$	-39…+24 dB, 1 dB step size.

Table A.70 — Coding of *bsGainSmoothAttackTimeSlow* field

Field	Encoding	Mnemonic	gainSmoothAttackTimeSlow in [ms]	Range
bsGainSmoothAttackTimeSlow	μ 8 bits	uimsbf	$\tau_{attack,slow} = \mu 5$	0…1275 ms, 5 ms step size.

Table A.71 — Coding of *bsGainSmoothReleaseTimeSlow* field

Field	Encoding	Mnemonic	gainSmoothReleaseTimeSlow in [ms]	Range
bsGainSmoothReleaseTimeSlow	μ 8 bits	uimsbf	$\tau_{release,slow} = \mu 40$	0…10200 ms, 40 ms step size.

Table A.72 — Coding of *bsGainSmoothAttackTimeFast* field

Field	Encoding	Mnemonic	gainSmoothAttackTimeFast in [ms]	Range
bsGainSmoothAttackTimeFast	μ 8 bits	uimsbf	$\tau_{attack,fast} = \mu 5$	0…1275 ms, 5 ms step size.

Table A.73 — Coding of *bsGainSmoothReleaseTimeFast* field

Field	Encoding	Mnemonic	gainSmoothReleaseTimeFast in [ms]	Range
bsGainSmoothReleaseTimeFast	μ 8 bits	uimsbf	$\tau_{release,fast} = \mu 20$	0…5100 ms, 20 ms step size.

Table A.74 — Coding of *bsGainSmoothAttackThreshold* field

Field	Encoding	Mnemonic	gainSmoothAttackThreshold in [dB]	Range
bsGainSmoothAttackThreshold	μ 5 bits	uimsbf	$Thr_{attack,fast} = \begin{cases} \mu; \text{ if } \mu \neq 31 \\ \infty; \text{ else} \end{cases}$	0…30 dB and ∞, 1 dB step size.

Table A.75 — Coding of *bsGainSmoothReleaseThreshold* field

Field	Encoding	Mnemonic	gainSmoothReleaseThreshold in [dB]	Range
bsGainSmoothReleaseThreshold	μ 5 bits	uimsbf	$Thr_{release,fast} = \begin{cases} \mu; \text{ if } \mu \neq 31 \\ \infty; \text{ else} \end{cases}$	0...30 dB and ∞, 1 dB step size.

Table A.76 — Coding of *bsGainSmoothHoldOff* field

Field	Encoding	Mnemonic	gainSmoothHoldOff	Range
bsGainSmoothHoldOff	μ 7 bits	uimsbf	$C_{holdOff} = \mu$	0...127.

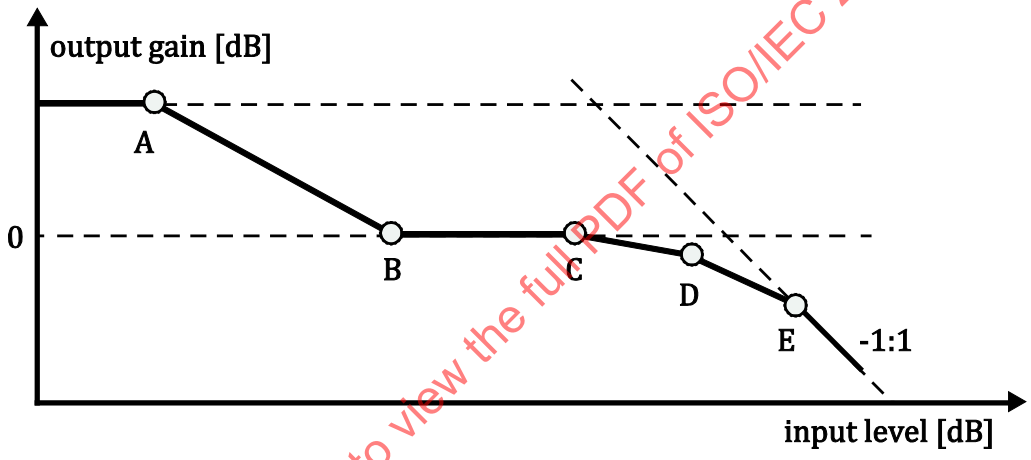


Figure A.1 — Illustration of DRC curve parametrization for five curve nodes (*parametricDrcType*==0x0)

Table A.77 — DRC gain smoothing time constants and look-ahead delay for drcCharacteristic index 7...11 (*parametricDrcType*==0x0)

Parameter	Default	drcCharacteristic index				
		7	8	9	10	11
parametricDrcLookAhead [ms]	10	10	10	10	10	10
gainSmoothAttackTimeSlow [ms]	100	100	100	100	100	100
gainSmoothReleaseTimeSlow [ms]	3 000	3 000	3 000	3 000	10 000	1 000
gainSmoothAttackTimeFast [ms]	10	10	10	10	10	10
gainSmoothReleaseTimeFast [ms]	1 000	1 000	1 000	1 000	1 000	200
gainSmoothAttackThreshold [dB]	15	15	15	15	15	10
gainSmoothReleaseThreshold [dB]	20	20	20	20	20	10
gainSmoothHoldOff	10	10	10	10	10	10

A.6.13 Coded metadata in parametricDrcTypeLimiter()

Table A.78 — Coding of metadata in parametricDrcTypeLimiter()

Metadata field	Description
bsParametricLimThreshold	A field that signals the limiting threshold for a parametric DRC instance of type PARAM_DRC_TYPE_LIM. See Table A.79. If not present, the default value is -1 dBFS. The limiting threshold relates to the signal after application of *loudnessNormalizationGainDb*.
parametricLimAttackTime	A field that signals the attack time for a parametric DRC instance of type PARAM_DRC_TYPE_LIM. This value is defined by the *bsParametricDrcLookAhead* field in the parametricDrcInstructions() payload. If not present, the default value is 5 ms.
bsParametricLimReleaseTime	A field that signals the release time for a parametric DRC instance of type PARAM_DRC_TYPE_LIM. See Table A.80. If not present, the default value is 50 ms.

Table A.79 — Coding of *bsParametricLimThreshold* field

Field	Encoding	Mnemonic	parametricLimThreshold in [dBFS]	Range
bsParametricLimThreshold	μ 8 bits	uimsbf	$Thr_{\text{lim}} = -\mu 2^{-3}$	-31.875 … 0 dBFS, 0.125 dB step size

Table A.80 — Coding of *bsParametricLimReleaseTime* field

Field	Encoding	Mnemonic	parametricLimReleaseTime in [ms]	Range
bsParametricLimReleaseTime	μ 8 bits	uimsbf	$\tau_{release,\text{lim}} = \mu 10$	0…2550 ms, 10 ms step size

A.6.14 Coded metadata in loudEqInstructions()

Table A.81 — Metadata in loudEqInstructions()

Metadata field	Description
loudEqSetId	Unique identifier for this LEQ set.
drcLocation	Location of associated DRC gain sequences (see Table 2).
downmixIdPresent	A flag that has a value of 1 if downmix IDs are present. A value of 0 indicates that the LEQ is only applicable to the base layout.
downmixId	Index to identify a downmix that is associated with this LEQ set. A value of 0 indicates that the LEQ can be applied to the base layout. A value of 0×7F indicates that the LEQ can be applied to any layout.
additionalDownmixIdPresent	A flag that has a value of 1 if additional downmix IDs are present.
additionalDownmixIdCount	Number of additional downmix IDs this LEQ set can be applied to.

Metadata field	Description
additionalDownmixId	Additional downmix IDs connected with this *loudEqSetId*. This field may be used to declare additional target layouts for which this LEQ set is applicable. This field has the same meaning as the *downmixId* field above.
drcSetIdPresent	A flag that has a value of 1 if drcSet IDs are present. A value of 0 indicates that the LEQ is only applicable if no DRC is applied.
drcSetId	Index to identify a DRC set that is associated with this LEQ set. A value of 0 indicates that the LEQ can be applied when no DRC is used. A value of 0×3F refers to any DRC including no DRC.
additionalDrcSetIdPresent	A flag that has a value of 1 if additional drcSet IDs are present.
additionalDrcSetIdCount	Number of additional drcSet IDs connected with this LEQ set.
additionalDrcSetId	Additional drcSet IDs connected with this LEQ set. This field may be used to declare additional DRC sets for which this LEQ set is applicable. The meaning of this field is identical to the *drcSetId* field above.
eqSetIdPresent	A flag that has a value of 1 if eqSet IDs are present. A value of 0 indicates that the LEQ is only applicable if no EQ is applied.
eqSetId	Index to identify an EQ set that is associated with this LEQ set. A value of 0 indicates that the LEQ can be applied when no EQ is used. A value of 0x3F refers to any EQ including no EQ.
additionalEqSetIdPresent	A flag that has a value of 1 if additional eqSet IDs are present.
additionalEqSetIdCount	Number of additional eqSet IDs connected with this LEQ set.
additionalEqSetId	Additional eqSet IDs connected with this LEQ set. This field may be used to declare additional EQ sets for which this LEQ set is applicable. The meaning of this field is identical to the *eqSetId* field above.
loudnessAfterDrc	A flag that has a value of 1 if the loudness values derived from the associated gain sequences are measured after a DRC was applied. Otherwise, the value is 0 and the loudness values should be corrected by adding the applied DRC gain [dB].
loudnessAfterEq	A flag that has a value of 1 if the loudness values derived from the associated gain sequences are measured after an EQ was applied. Otherwise, the value is 0 and the loudness values should be corrected by adding the applied EQ gain [dB].
loudEqGainSequenceCount	The number of gain sequences associated with this LEQ set.
gainSequenceIndex	Index referring to a drcGainSequence() in uniDrcGain() payload.
drcCharacteristicFormatIsCICP	Flag indicating whether the index of the characteristic is based on ISO/IEC 23008-1, (1) or not (0).
drcCharacteristic	Index of DRC characteristic according to ISO/IEC 23091-3 (CICP). The value shall be non-zero and the characteristic shall be invertible, i.e. it shall have either a negative or positive slope.
drcCharacteristicLeftIndex, drcCharacteristicRightIndex	Index of DRC characteristic associated with the gain sequence referring to *characteristicLeftIndex* and *characteristicRightIndex* in drcCoefficientsUniDrcV1(). The characteristic shall be invertible, i.e. it shall have either a negative or positive slope.
frequencyRangeIndex	Specifies the frequency range associated with the gain sequence according to Table A.82. The associated reference level specifies which acoustic signal level of a sinusoid corresponds to the level values derived from the gain sequence.

Metadata field	Description
bsLoudEqScaling	Contains an encoded linear scaling coefficient for the loudness values [dB] according to Table A.83.
bsLoudEqOffset	Contains an encoded offset value [dB] for the loudness values [dB] according to Table A.84. The offset is applied after scaling, if applicable.

Table A.82 — Coding of *frequencyRangeIndex* in *loudEqInstructions*()

frequencyRangeIndex	Frequency range	Reference level
0	Low frequencies: 20 Hz to 250 Hz	100 Hz sinusoid with 100 dB SPL at default listener position results in 0 dB after applying the inverse DRC characteristic.
1	High frequencies: 10 kHz and higher	12 kHz sinusoid with 100 dB SPL at default listener position results in 0 dB after applying the inverse DRC characteristic.
(All remaining values are reserved for future use)	*For future use*	*For future use*

Table A.83 — Coding of *bsLoudEqScaling* in loudEqInstructions()

bsLoudEqScaling	Loudness scaling coefficient	Range
μ	$2^{-0.5\mu}$	0.0884 ... 1.0

Table A.84 — Coding of *bsLoudEqOffset* in loudEqInstructions()

bsLoudEqOffset	Loudness offset [dB]	Range
μ	$1.5\mu - 16$	-16 ... 30.5 dB, 1.5 dB step size

A.6.15 Coding of payloads for equalization

Table A.85 — Coding of fields in eqCoefficients() payload

Field	Encoding	Description
bsEqDelayMaxPresent	Flag {0,1}	If 1, *bsEqDelayMax* value is present
bsEqDelayMax	Delay value, see Table A.86	Maximum EQ delay of any applicable EQ set combination in the stream
uniqueFilterBlockCount	Count [0,63]	Number of unique filter blocks
filterElementCount	Count [0,63]	Number of filter elements in a block
filterElementIndex	Index [0,63]	Index refers to filter element definition
filterElementGainPresent	Flag {0,1}	If 1, filter element gain value is present
bsFilterElementGain	Gain value, see Table A.92	Filter element gain value
uniqueTdFilterElementCount	Count [0,63]	Number of unique filter elements
eqFilterFormat	Format identifier [0,1]	0: Pole/zero, 1: FIR coefficient

Field	Encoding	Description
bsRealZeroRadiusOneCount	Count=2*bsRealZeroRadiusOneCount {0,2,...,14}	Number of real zeros with a radius of 1
realZeroCount	Count [0,63]	Number of real zeros
genericZeroCount	Count [0,63]	Number of generic zeros
realPoleCount	Count [0,15]	Number of real poles
complexPoleCount	Count [0,15]	Number of complex poles
zeroSign, poleSign	Sign of real zero / pole location {0,1}	If 1, sign is negative
bsZeroRadius, bsPoleRadius	Radius value, see Table A.95	Radius of zero/pole location in z-plane
bsZeroAngle, bsPoleAngle	Angle value, see Table A.96	Angle of zero/pole location in z-plane
firFilterOrder	Order value [0, 127]	Order *M* of FIR filter
firSymmetry	Symmetry value {0,1}	0: symmetric, 1: anti-symmetric
bsFirCoefficient	Coefficient value, see Table A.93	Value of FIR coefficient. Only the first half of the impulse response is transmitted in natural order
uniqueEqSubbandGainsCount	Count [0,63]	Number of sub-band gain vectors
eqSubbandGainFormat	Format value, see Table A.90	Specifies format of sub-band gain vector
eqSubbandGainRepresentation	Flag {0,1}	Specifies the representation of sub-band gain values. 0: value per band, 1: spline
bsEqGainCount	Count=bsEqGainCount+1 [1,256]	Number of sub-band gain values in a vector
bsEqSubbandGain	Gain value, see Table A.91	Sub-band gain value

Table A.86 — Coding of *bsEqDelayMax* field

Field	Encoding	Mnemonic	eqDelayMax in intervals at audio sample rate	Range
bsEqDelayMax	$\{\mu, v\}$ {5 bits, 3 bits}	{uimsbf, uimsbf}	$16\mu 2^{v}$	0...63488, variable step size

Table A.87 — Coding of fields in eqSubbandGainSpline() payload

Field	Encoding	Description
bsEqNodeCount	See Table A.97.	Node count of EQ gain spline.
eqSlopeCode	See Table A.98.	Slope steepness at spline node in [dB/octave].
eqFreqDeltaCode	See Table A.99.	Differential sub-band index representation of spline node.
eqGainInitialCode	See Table A.100.	EQ gain for the spline node in the lowest sub-band.
eqGainDeltaCode	See Table A.101.	Differential representation of EQ gain at the corresponding spline node.

Table A.88 — Coding of fields in eqInstructions() payload

Field	Encoding	Description
eqSetId	Index [0,63]	Identifier for the EQ set defined in eqInstructions(). Values of 0 and 0x3F are reserved.
eqSetComplexityLevel	[0,15] see 6.9	Value that indicates the computational complexity level of the associated EQ set per audio channel. The complexity is used to determine if a decoder is capable of applying the EQ set.
downmixIdPresent	Flag {0,1}	If 1, downmix IDs are present.
downmixId, additionalDownmixId	Index [0,127]	Identifies all downmix configurations by their ID that this EQ can be combined with. A value of 0 refers to the base layout, 0×7F refers to any downmix and base layout.
additionalDownmixIdPresent	Flag {0,1}	If 1, additional downmix IDs are present. Note that in this case only one *eqChannelGroup* is permitted if *eqApplyToDownmix* is 1.
additionalDownmixIdCount	Count [0,127]	Number of additional downmix IDs.
eqApplyToDownmix	Flag {0,1}	If 1, the EQ set is applied to the downmix. Otherwise it is applied to the base layout.
drcSetId, additionalDrcSetId	Index [0,63]	Identifies a DRC set that can be applied in combination with this EQ, unless this is a dependent EQ. A value of 0 indicates that it is permitted to apply the EQ without DRC. A value of 0×3F refers to any DRC including no DRC.
additionalDrcSetIdPresent	Flag {0,1}	If 1, additional drcSet IDs are present.
additionalDrcSetIdCount	Count [0,63]	Number of additional drcSet IDs.
eqSetPurpose	Bit field, see Table A.89	Defines the purpose(s) of the EQ.
dependsOnEqSetPresent	Flag {0,1}	If 1, the EQ set shall be applied in combination with a second EQ set. The second EQ set shall be applied at the opposite side of the downmixer.
dependsOnEqSet	Index [0,63]	Index of the EQ set that shall be combined with this EQ set. The combined EQs can be applied with any DRC set that is specified by a *drcSetId* in this EQ set.
noIndependentEqUse	Flag {0,1}	If 1, the EQ set can only be used in combination with a scond EQ set.
eqChannelGroup	Index [1,127]	Index of EQ channel group that the channel belongs to.
tdFilterCascadePresent	Flag {0,1}	Indicates if a time-domain filter cascade is defined.
eqCascadeGainPresent	Flag {0,1}	Indicates if a filter cascade gain value is present.
bsEqCascadeGain	Gain value, see Table A.92	Filter cascade gain value.
filterBlockCount	Count [0,15]	Number of filter blocks contained in this cascade.
filterBlockIndex	Index [0,127]	Index refers to definition of filter block in eqCoefficients().
eqPhaseAlignmentPresent	Flag {0,1}	If 1, indicates that phase alignment information is present.
bsEqPhaseAlignment	Flag {0,1}	If 1, indicates that the corresponding EQ channel groups are phase aligned.

Field	Encoding	Description
subbandGainsPresent	Flag {0,1}	If 1, indicates that gain values for sub-band gains are present.
subbandGainsIndex	Index [0,63]	Index refers to sub-band gain vector in eqCoefficients().
eqTransitionDurationPresent	Flag {0,1}	If 1, indicates that a transition duration value is present.
bsEqTransitionDuration	Time value, see Table A.94	Transition duration for crossfading from the output of the previous EQ set to the current EQ set.

Table A.89 — Coding of *eqSetPurpose* field (multiple bits can be set)

Bit position	EQ purpose	Description (valid if bit is set)
1 (LSB)	Default EQ	For generic use.
2	Large room	For playback in rooms with a volume significantly larger than a typical living room.
3	Small space	For playback in rooms with a volume significantly smaller than a typical living room.
4	Average room	For playback in rooms with a volume of a typical living room.
5	Car cabin	For playback in a passenger car.
6	Headphones	For playback with headphones.
7	Late night	For playback with reduced exposure for others nearby, such as someone in an adjacent room, to minimize potential disturbance.
Remaining bits reserved	*reserved*	—

Table A.90 — Coding of *eqSubbandGainFormat* field

Encoding	Name	Description
0×0	*Reserved*	—
0×1	GAINFORMAT_QMF32	Gain values correspond to the center frequencies of a 32 band QMF filter bank as defined in Reference [9].
0×2	GAINFORMAT_QMFHYBRID39	Gain values correspond to the center frequencies of a 39 band hybrid QMF filter bank as defined in Reference [9].
0×3	GAINFORMAT_QMF64	Gain values correspond to the center frequencies of a 64 band QMF filter bank as defined in Reference [9].
0×4	GAINFORMAT_QMFHYBRID71	Gain values correspond to the center frequencies of a 71 band hybrid QMF filter bank as defined in Reference [9].
0×5	GAINFORMAT_QMF128	Gain values correspond to the center frequencies of a 128 band QMF filter bank as defined in Reference [9].
0×6	GAINFORMAT_QMFHYBRID135	Gain values correspond to the center frequencies of a 135 band hybrid QMF filter bank as defined in Reference [9].
0×7	GAINFORMAT_UNIFORM	Gain values correspond to sub-band center frequencies where all sub-bands cover the entire audio spectrum and have identical bandwidth and no overlap.

Encoding	Name	Description
Remaining values are reserved	*reserved*	—

Table A.91 — Coding of *bsEqSubbandGain*

Encoding	Size	Mnemonic	Subband gain [dB]	Range
{σ, μ}	{1 bit, 8 bits}	{uimsbf, uimsbf}	$g = (-1)^{\sigma}\ \mu 2^{-3}$	-31.875...31.875 dB, 0.125 dB step size

Table A.92 — Coding of *bsEqCascadeGain*, *bsFilterElementGain*

Encoding	Size	Mnemonic	Gain [dB]	Range
μ	10 bits	uimsbf	$g = \mu 2^{-3} - 96$	-96...31.875 dB, 0.125 dB step size

Table A.93 — Coding of *bsFirCoefficient*

Encoding	Size	Mnemonic	Coefficient	Range
{σ, μ}	{1 bit, 10 bits}	{uimsbf, uimsbf}	$b = (-1)^{\sigma}\ 10^{-0.05\mu 2^{-4}}$	signed linear values, -63.9375...0 dB, 0.0625 dB step size

Table A.94 — Coding of *bsEqTransitionDuration*

Encoding	Size	Mnemonic	eqTransitionDuration [s]	Range
μ	5 bits	uimsbf	$\tau = 0.001 \cdot 2^{2+\mu/4}$	4...861 ms

Table A.95 — Coding of *bsPoleRadius* and *bsZeroRadius*

bsPoleRadius, bsZeroRadius	ρ	bsPoleRadius, bsZeroRadius	ρ
0	0.00000000E+00	64	9.41318572E-02
1	7.57409621E-11	65	9.98248383E-02
2	7.47451079E-09	66	1.05744988E-01
3	7.37623509E-08	67	1.11896060E-01
4	3.37872933E-07	68	1.18281692E-01
5	1.05439995E-06	69	1.24905407E-01
6	2.61370951E-06	70	1.31770656E-01
7	5.55702854E-06	71	1.38880774E-01
8	1.05878771E-05	72	1.46238968E-01
9	1.85806475E-05	73	1.53848350E-01
10	3.05868707E-05	74	1.61711931E-01

bsPoleRadius, bsZeroRadius	ρ	bsPoleRadius, bsZeroRadius	ρ
11	4.78395414E-05	75	1.69832602E-01
12	7.17558214E-05	76	1.78213134E-01
13	1.03938342E-04	77	1.86856180E-01
14	1.46175269E-04	78	1.95764288E-01
15	2.00439375E-04	79	2.04939872E-01
16	2.68886099E-04	80	2.14385241E-01
17	3.53850890E-04	81	2.24102572E-01
18	4.57845890E-04	82	2.34093949E-01
19	5.83555840E-04	83	2.44361281E-01
20	7.33833469E-04	84	2.54906416E-01
21	9.11694835E-04	85	2.65731007E-01
22	1.12031354E-03	86	2.76836663E-01
23	1.36301492E-03	87	2.88224846E-01
24	1.64327072E-03	88	2.99896836E-01
25	1.96469179E-03	89	3.11853856E-01
26	2.33102194E-03	90	3.24096978E-01
27	2.74613220E-03	91	3.36627185E-01
28	3.21401190E-03	92	3.49445283E-01
29	3.73876374E-03	93	3.62551987E-01
30	4.32459544E-03	94	3.75947863E-01
31	4.97581391E-03	95	3.89633417E-01
32	5.69681637E-03	96	4.03608948E-01
33	6.49208482E-03	97	4.17874694E-01
34	7.36617809E-03	98	4.32430804E-01
35	8.32372531E-03	99	4.47277188E-01
36	9.36941616E-03	100	4.62413728E-01
37	1.05079999E-02	101	4.77840215E-01
38	1.17442720E-02	102	4.93556231E-01
39	1.30830696E-02	103	5.09561300E-01
40	1.45292655E-02	104	5.25854886E-01
41	1.60877611E-02	105	5.42436182E-01
42	1.77634824E-02	106	5.59304416E-01
43	1.95613634E-02	107	5.76458573E-01
44	2.14863531E-02	108	5.93897760E-01
45	2.35434026E-02	109	6.11620665E-01
46	2.57374570E-02	110	6.29626155E-01

bsPoleRadius, bsZeroRadius	ρ	bsPoleRadius, bsZeroRadius	ρ
47	2.80734543E-02	111	6.47912800E-01
48	3.05563174E-02	112	6.66479111E-01
49	3.31909470E-02	113	6.85323536E-01
50	3.59822176E-02	114	7.04444408E-01
51	3.89349759E-02	115	7.23839939E-01
52	4.20540236E-02	116	7.43508339E-01
53	4.53441292E-02	117	7.63447523E-01
54	4.88100089E-02	118	7.83655465E-01
55	5.24563305E-02	119	8.04130018E-01
56	5.62877022E-02	120	8.24868977E-01
57	6.03086725E-02	121	8.45869958E-01
58	6.45237267E-02	122	8.67130578E-01
59	6.89372867E-02	123	8.88648331E-01
60	7.35536888E-02	124	9.10420537E-01
61	7.83772022E-02	125	9.32444632E-01
62	8.34120139E-02	126	9.54717815E-01
63	8.86622295E-02	127	9.77237225E-01

Table A.96 — Coding of *bsPoleAngle* and *bsZeroAngle*

bsPoleAngle, bsZeroAngle	Normalized angle $\alpha = \frac{\theta}{\pi}$	bsPoleAngle, bsZeroAngle	Normalized angle $\alpha = \frac{\theta}{\pi}$
0	0.00000000E+00	64	2.62780130E-02
1	6.90533966E-04	65	2.78405849E-02
2	7.31595252E-04	66	2.94960723E-02
3	7.75098170E-04	67	3.12500000E-02
4	8.21187906E-04	68	3.31082217E-02
5	8.70018279E-04	69	3.50769390E-02
6	9.21752258E-04	70	3.71627223E-02
7	9.76562500E-04	71	3.93725328E-02
8	1.03463193E-03	72	4.17137454E-02
9	1.09615434E-03	73	4.41941738E-02
10	1.16133507E-03	74	4.68220962E-02
11	1.23039165E-03	75	4.96062829E-02
12	1.30355455E-03	76	5.25560260E-02
13	1.38106793E-03	77	5.56811699E-02

bsPoleAngle, bsZeroAngle	Normalized angle $\alpha = \frac{\theta}{\pi}$	bsPoleAngle, bsZeroAngle	Normalized angle $\alpha = \frac{\theta}{\pi}$
14	1.46319050E-03	78	5.89921445E-02
15	1.55019634E-03	79	6.25000000E-02
16	1.64237581E-03	80	6.62164434E-02
17	1.74003656E-03	81	7.01538780E-02
18	1.84350452E-03	82	7.43254447E-02
19	1.95312500E-03	83	7.87450656E-02
20	2.06926386E-03	84	8.34274909E-02
21	2.19230869E-03	85	8.83883476E-02
22	2.32267015E-03	86	9.36441923E-02
23	2.46078330E-03	87	9.92125657E-02
24	2.60710909E-03	88	1.05112052E-01
25	2.76213586E-03	89	1.11362340E-01
26	2.92638101E-03	90	1.17984289E-01
27	3.10039268E-03	91	1.25000000E-01
28	3.28475162E-03	92	1.32432887E-01
29	3.48007312E-03	93	1.40307756E-01
30	3.68700903E-03	94	1.48650889E-01
31	3.90625000E-03	95	1.57490131E-01
32	4.13852771E-03	96	1.66854982E-01
33	4.38461738E-03	97	1.76776695E-01
34	4.64534029E-03	98	1.87288385E-01
35	4.92156660E-03	99	1.98425131E-01
36	5.21421818E-03	100	2.10224104E-01
37	5.52427173E-03	101	2.22724680E-01
38	5.85276202E-03	102	2.35968578E-01
39	6.20078536E-03	103	2.50000000E-01
40	6.56950324E-03	104	2.64865774E-01
41	6.96014624E-03	105	2.80615512E-01
42	7.37401807E-03	106	2.97301779E-01
43	7.81250000E-03	107	3.14980262E-01
44	8.27705542E-03	108	3.33709964E-01
45	8.76923475E-03	109	3.53553391E-01
46	9.29068059E-03	110	3.74576769E-01
47	9.84313320E-03	111	3.96850263E-01
48	1.04284364E-02	112	4.20448208E-01

bsPoleAngle, bsZeroAngle	Normalized angle $\alpha = \frac{\theta}{\pi}$	bsPoleAngle, bsZeroAngle	Normalized angle $\alpha = \frac{\theta}{\pi}$
49	1.10485435E-02	113	4.45449359E-01
50	1.17055240E-02	114	4.71937156E-01
51	1.24015707E-02	115	5.00000000E-01
52	1.31390065E-02	116	5.29731547E-01
53	1.39202925E-02	117	5.61231024E-01
54	1.47480361E-02	118	5.94603558E-01
55	1.56250000E-02	119	6.29960525E-01
56	1.65541108E-02	120	6.67419927E-01
57	1.75384695E-02	121	7.07106781E-01
58	1.85813612E-02	122	7.49153538E-01
59	1.96862664E-02	123	7.93700526E-01
60	2.08568727E-02	124	8.40896415E-01
61	2.20970869E-02	125	8.90898718E-01
62	2.34110481E-02	126	9.43874313E-01
63	2.48031414E-02	127	1.00000000E+00

Table A.97 — Coding of node count of EQ gain spline (*bsEqNodeCount*)

Encoding	Size	Mnemonic	eqFreqDelta	Range
μ	5 bits	uimsbf	$\mu + 2$	2...33

Table A.98 — Coding of EQ slope steepness (*eqSlopeCode*)

Encoding [hex] (eqSlopeCode)	Encoding size [bits]	EQ slope steepness [dB/octave]
0×1	1	0.0
0×00	5	-32.0
0×01	5	-24.0
0×02	5	-18.0
0×03	5	-12.0
0×04	5	-7.0
0×05	5	-4.0
0×06	5	-2.0
0×07	5	-1.0
0×08	5	1.0
0×09	5	2.0
0×0A	5	4.0

Encoding [hex] (eqSlopeCode)	Encoding size [bits]	EQ slope steepness [dB/octave]
0×0B	5	7.0
0×0C	5	12.0
0×0D	5	18.0
0×0E	5	24.0
0×0F	5	32.0

Table A.99 — Coding of EQ frequency difference (*eqFreqDeltaCode*)

Encoding	Size	Mnemonic	eqFreqDelta	Range
μ	4 bits	uimsbf	$\mu + 1$	1...16

Table A.100 — Coding of initial sub-band EQ gain value (*eqGainInitialCode*)

Encoding	Size	Mnemonic	Gain [dB]	Range
{0×0,μ}	{2 bits, 5 bits}	bslbf	0.5μ-8.0	-8...7.5
{0×1,μ}	{2 bits, 4 bits}	{bslbf, uimsbf}	μ -16; if μ<8 μ; else	-16...-9, 8...15
{0×2,μ}	{2 bits, 4 bits}	{bslbf, uimsbf}	2μ-32; if μ<8 2μ; else	-32...-18, 16...30
{0×3,μ}	{2 bits, 3 bits}	{bslbf, uimsbf}	4μ-64	-64...-36

Table A.101 — Coding of EQ gain differences (*eqGainDeltaCode*)

eqGainDeltaCode	eqGainDelta [dB]	eqGainDeltaCode	eqGainDelta [dB]
0	-22.0	16	0.5
1	-16.0	17	1.0
2	-13.0	18	1.5
3	-11.0	19	2.0
4	-9.0	20	2.5
5	-7.0	21	3.0
6	-6.0	22	4.0
7	-5.0	23	5.0
8	-4.0	24	6.0
9	-3.0	25	7.0
10	-2.5	26	9.0
11	-2.0	27	11.0
12	-1.5	28	13.0
13	-1.0	29	16.0
14	-0.5	30	22.0
15	0.0	31	32.0

A.6.16 Summary of supported control parameters supplied by host

Table A.102 — Summary of all supported control parameters for loudness normalization supplied by host

Identifier	Default value	Unit	Description
targetLoudness	—	dB	Desired output loudness.
albumMode	FALSE	Boolean	See 6.3.6.
loudnessDeviationMax	63	dB	Permitted deviation from the exact loudness normalization.
loudnessMeasurementMethod	Program Loudness	—	Requested method (see Table 47).
loudnessMeasurementSystem	ITU-R BS.1770-4	—	Requested system (see Table 48)
loudnessMeasurementPreProc	No pre-processing	—	Requested pre-processing (see Table 49).
deviceCutOffFrequency	20	Hz	Lower acoustic frequency limit of playback system response.
loudnessNormalizationGainDbMax	∞	dB	Maximum permitted normalization gain.
loudnessNormalizationGainModificationDb	0	dB	Normalization gain modification.
outputPeakLevelMax	0, (6, if *peakLimiterPresent*)	dB	Maximum permitted peak value.
peakLimiterPresent	FALSE	Boolean	Indicates if a peak limiter is present in the playback chain.

Table A.103 — Summary of all supported control parameters for dynamic range compression supplied by host

Identifier	Default value	Unit	Description
dynamicRangeMeasurement	—	—	See Table 13.
dynamicRangeMeasurementValue	—	dB	The requested value corresponding to *dynamicRangeMeasurement.*
dynamicRangeMeasurementValueMin, dynamicRangeMeasurementValueMax	—	dB	The requested range corresponding to *dynamicRangeMeasurement.*
drcCharacteristic	—	—	See Table E.4 and Table E.5 .
drcEffectType (desired/fallback)	—	—	See Table 12.
compress	1	—	Scaling factor for negative DRC gains [dB].
boost	1	—	Scaling factor for positive DRC gains [dB].
drcCharacteristicTarget	0	—	Target DRC characteristic. See Table E.4 and Table E.5.

Table A.104 — Summary of additional control parameters supplied by host

Identifier	Default value	Unit	Description
loudnessEqRequest	0×0	—	Requested loudness EQ type.
sensitivity	90.0	dB SPL	Sensitivity of the playback system (see D.2.10).
playbackGain	-15.0	dB	Playback gain of the playback system (see D.2.10).
eqSetPurposeRequest	0×0	—	Requested EQ set purpose.

Annex B
(normative)

External interface to DRC tool

B.1 Description

This annex describes a normative decoder interface to the DRC tool for external parameter access in various applications. A non-exhaustive list of example use cases is given below:

— Control of DRC tool from a higher system layer (e.g. all kinds of graphical user interfaces, which allow manipulating the DRC effect, the target loudness or others).

— Adjustment of DRC tool parameters to changed playback conditions (e.g. switch from loudspeaker to headphone playback).

— Remote control of DRC tool from a different device than it is implemented on.

— Adjustment of loudness normalization gain based on live measurements or other parameters available at the decoder.

The syntax and semantics of the interface are defined in B.2 and B.3, respectively. Decoders shall support (i.e. parse) received interface streams and normatively act on them. In local environments, the support of alternative interfaces is optionally permitted.

Although the interface is optional, it is recommended to use it whenever possible.

The syntax as defined in Table B.1 includes an optional interface signature, which can be used for various tasks. Some examples are listed below:

— Allow parameter changes only for authorized signatures.

— Allow hardware related parameter changes only to specific signatures.

— Allow loudness related parameter changes only to specific signatures.

— Define priorities for certain signatures if contradicting parameters are received.

B.2 Syntax

Various examples of syntax are provided in Tables B.1 - B.9.

Table B.1 — Syntax of uniDrcInterface() payload

Syntax	No. of bits	Mnemonic
uniDrcInterface()		
{		
uniDrcInterfaceSignaturePresent;	**1**	**bslbf**
if (uniDrcInterfaceSignaturePresent == 1) {		
uniDrcInterfaceSignatureType;	**8**	**uimsbf**

Syntax	No. of bits	Mnemonic
bsUniDrcInterfaceSignatureDataLength;	**8**	**uimsbf**
for (c = 0; c < bsUniDrcInterfaceSignatureDataLength+1; c++) {		
uniDrcInterfaceSignatureData[c];	**8**	**uimsbf**
}		
}		
systemInterfacePresent;	**1**	**bslbf**
if (systemInterfacePresent == 1) {		
systemInterface();		
}		
loudnessNormalizationControlInterfacePresent;	**1**	**bslbf**
if (loudnessNormalizationControlInterfacePresent == 1) {		
loudnessNormalizationControlInterface();		
}		
loudnessNormalizationParameterInterfacePresent;	**1**	**bslbf**
if (loudnessNormalizationParameterInterfacePresent == 1) {		
loudnessNormalizationParameterInterface();		
}		
dynamicRangeControlInterfacePresent;	**1**	**bslbf**
If (dynamicRangeControlInterfacePresent == 1) {		
dynamicRangeControlInterface();		
}		
dynamicRangeControlParameterInterfacePresent;	**1**	**bslbf**
If (dynamicRangeControlParameterInterfacePresent == 1) {		
dynamicRangeControlParameterInterface();		
}		
uniDrcInterfaceExtensionPresent;	**1**	**bslbf**
if (uniDrcInterfaceExtensionPresent == 1) {		
uniDrcInterfaceExtension();		
}		
}		

Table B.2 — Syntax of systemInterface() payload

Syntax	No. of bits	Mnemonic
systemInterface()		
{		
targetConfigRequestType;	**2**	**uimsbf**
if (targetConfigRequestType == 0) {		
numDownmixIdRequests;	**4**	**uimsbf**

Syntax	No. of bits	Mnemonic
for (d = 0; d < numDownmixIdRequests; d++) {		
downmixIdRequested[d];	**7**	**uimsbf**
}		
} else if (targetConfigRequestType == 1) {		
targetLayoutRequested;	**8**	**uimsbf**
} else if (targetConfigRequestType == 2) {		
bsTargetChannelCountRequested;	**7**	**uimsbf**
}		
}		

Table B.3 — Syntax of loudnessNormalizationControlInterface() payload

Syntax	No. of bits	Mnemonic
loudnessNormalizationControlInterface()		
{		
loudnessNormalizationOn;	**1**	**bslbf**
if (loudnessNormalizationOn == 1) {		
targetLoudness;	**12**	**uimsbf**
}		
}		

Table B.4 — Syntax of loudnessNormalizationParameterInterface() payload

Syntax	No. of bits	Mnemonic
loudnessNormalizationParameterInterface()		
{		
albumMode;	**1**	**bslbf**
peakLimiterPresent;	**1**	**bslbf**
changeLoudnessDeviationMax;	**1**	**bslbf**
if (changeLoudnessDeviationMax == 1) {		
loudnessDeviationMax;	**6**	**uimsbf**
}		
changeLoudnessMeasurementMethod;	**1**	**bslbf**
if (changeLoudnessMeasurementMethod == 1) {		
loudnessMeasurementMethod;	**3**	**uimsbf**
}		
changeLoudnessMeasurementSystem;	**1**	**bslbf**
if (changeLoudnessMeasurementSystem == 1) {		
loudnessMeasurementSystem;	**4**	**uimsbf**
}		